Norsk Regnesentral
NORWEGIAN COMPUTING CENTER

Note

CreolE — A pragmatic extension to *Creol*

Note no DART/05/09
Author Wolfgang Leister
Date 18th August 2009

Norsk Regnesentral
NORWEGIAN COMPUTING CENTER

Norwegian Computing Center

Norsk Regnesentral (Norwegian Computing Center, NR) is a private, independent, non-profit foundation established in 1952. NR carries out contract research and development projects in the areas of information and communication technology and applied statistical modeling. The clients are a broad range of industrial, commercial and public service organizations in the national as well as the international market. Our scientific and technical capabilities are further developed in co-operation with The Research Council of Norway and key customers. The results of our projects may take the form of reports, software, prototypes, and short courses. A proof of the confidence and appreciation our clients have for us is given by the fact that most of our new contracts are signed with previous customers.

Norsk Regnesentral
Norwegian Computing Center
Postboks 114, Blindern
NO-0314 Oslo, Norway

Besøksadresse
Office address
Gaustadalléen 23
NO-0373 Oslo, Norway

Telefon · telephone
(+47) 22 85 25 00
Telefaks · telefax
(+47) 22 69 76 60

Internett · internet
www.nr.no
E-post · e-mail
nr@nr.no

Title	***CreolE* — A pragmatic extension to *Creol***
Author	**Wolfgang Leister**
Quality assurance	Bjarte M. Østvold
Date	18th August 2009
Publication number	DART/05/09

Abstract

The present document describes extensions to the modelling language *Creol* which are implemented using the cpp pre-processor. The purpose of these extensions is to support constants, dependent compilation, and to provide convenience functions that make modelling easier.

Keywords	*CreolE*, *Creol*, CREDO, cpp
Target group	Developers of Creol models
Availability	Open
Project	CREDO
Project number	320362
Research field	Formal Methods
Number of pages	7

1 Introduction

While modelling biomedical sensor networks in *Creol* (Leister et al. (2009)) we recognised the need for a extensions of *Creol* (Kyas (2009)) from in order to make *Creol* more suited for large models. The extensions proposed here are based on the GNU cpp program (see `man cpp`). In the following we discuss these extensions using important best practice cases.

2 Best practice

The following sections contain the additional features of *CreolE*, and give a motivation for their use. *CreolE* stands for "**Creol E**xtended".

2.1 Named Constants

Standard *Creol* does not allow for named constants. However, in some occasions named constants result in better models in the sense of less modelling errors. For example, to model messages exchanged in a biomedical sensor network we need to model several message types, denoted by an Int. When the model gets large, it is more suitable to use named constants in order to avoid using the wrong constant.

As an example we show how to define named constants in the BSN model, with five different message types:

```
#define _mt_Payload  1
#define _mt_RREQ     2
#define _mt_RREP     3
#define _mt_RERR     4
#define _mt_Flooding 99
```

We use the named constants as shown in the following snippet:

```
var theMessageType: Int;
theMessageType :=get(tmsg,"MessageType");
if  theMessageType = _mt_Payload then
    INC(numPayloadSent);
    processOutgoingPayloadMessage(tmsg;success);
End;
```

2.2 Named Types

In the BSN model we use the data type `Map[String,Int]` to model messages. However, in large models it is impractical to write this type definition at all occurrences due to maintenance consts, e.g., when changing the representation. Therefore, we introduce named types as shown in the following snippet:

```
#define AMessage Map[String,Int]
```

```
...
  op processMessage(in tpm: AMessage; out success: Bool) ==
    var theMessage: AMessage;
    theMessage := tpm;
     ...
```

2.3 Conditional Inclusion

The cpp allows for conditional inclusion or exclusion of parts of the code. This is necessary to maintain the code of large models. Using comments to switch on/off parts of the code is rather impractical, especially when making larger changes such as re-defining a data type throughout the entire model. Therefore the standard use of `#if` and `#ifdef` is supported.

Note that switching code on/off can lead to the so-called semicolon-problem in *Creol*, which is illustrated in the following snippet which will cause a compilation failure when `_USE_COUNTER` is set to 0.

```
#define _USE_COUNTER 1
if existRouteToDst then
   await network.singlecast(tpm,nexthop;scresult);
#if _USE_COUNTER
   counter := counter + 1
#endif
end
```

In order to solve this problem we use the skip command and the extended `End` and `Else` statements.

2.4 Extended `End` and `Else` statements

The extended `End` and `Else` statements, written with an initial capital letter, are used to avoid the semicolon-problem of *Creol*, which occurs especially for conditional compilation. In practice, a skip statement is inserted before the command in minuscules. Therefore, using the extended statements you always must set a semicolon at the end of the previous *Creol* command. The use of the extended statements is shown in the following snippet.

```
#define _USE_COUNTER 1

if existRouteToDst then
   await network.singlecast(tpm,nexthop;scresult);
#if _USE_COUNTER
   aCounter := aCounter + 1;
#endif
Else
   bCounter := bCounter +1;
End
```

Note that the extended `Else` and `End` statements also allow for empty branches, as the following compilable code snippet shows:

```
#define _USE_COUNTER 0

if existRouteToDst then
#if _USE_COUNTER
   aCounter := aCounter + 1;
#endif
Else
   bCounter := bCounter +1;
End
```

2.5 File Inclusion

The `cpp` allows for including other *CreolE* files into the code using `#include`, which is also supported in *CreolE*.

2.6 The Macros `DEC`, `INC`, `INSERT`, and `REMOVE`

When using counters *Creol* offers an assignment as follows:

```
counter := counter +1;
```

Quite frequently, when copying and modifying code, one of the two variables when increasing a counter is forgotten to be changed by accident, resulting in strange results, and tedious debugging of the model. In order to avoid this we introduce the macros `INC(v)` and `DEC(v)`, as well as `INSERT(m,e,c)` and `REMOVE(m,e)` for inserting or replacing, and removing elements in maps, sets, etc.

3 How to Use *CreolE*

The *CreolE* extension needs the GNU cpp, the file `std.creole`, and changes in the makefile. The latter two are shown in the following.

3.1 The File `std.creole`

The *CreolE* extension contains the file `std.creole` which provides the necessary macros.

```
#ifndef _STD_CREOLE
#define _STD_CREOLE 1
#define INC(A) A := A + 1
#define DEC(A) A :=A - 1
#define INSERT(A,E,C) A :=insert(A,E,C)
#define REMOVE(A,E) A :=remove(A,E)
#define Else skip else
#define End skip end
#endif
```

In the *CreolE* code you must include the file `std.creole` by the following statement:

```
#include <std.creole>
```

3.2 Changes to the Makefile

When using *CreolE* the Makefile should include the following extra definitions. Note that the first first line, using the option `-P` should be used for older *Creol* compilers prior to *Creol* version 0.0n.

```
#CPP = cpp -I . -P -C
CPP = cpp -I . -C

%.creol : %.creole
        $(CPP) $< -o $@
```

4 Experiences with *CreolE*

CreolE was used to port a *Creol* model of over 1000 lines of code (Leister et al. (2009)) from one data type using classes for messages to another data type using maps. To do this the porting needed to be done on parts of the code at a time. We chose to port each message type at a time, which turned out to be the right portioning. Using *CreolE* porting the model was done in about one working day. Additionally, *CreolE* now allows us to include several test scenarios that can be switched on/off, or be included from files.

In order to align the line numbers in error messages to the .creole files the creol-compiler has been extended to interpret line markers generated by `cpp`, which is supported by compiler version 0.0n and newer.

We hope that *CreolE* is found useful by developers of *Creol* models. Note that *CreolE* is work in progress, and new features might be introduced as experiences with modelling in *Creol* and *CreolE* progress.

References

Kyas, M. (2009). Creoltools. Available for download at http://heim.ifi.uio.no/~kyas/creoltools/.

Leister, W., Liang, X., Stam, A., Klüppelholz, S., and Jaghoori, M. (2009). Credo deliverable 6.3: Final modelling. 1. 03. 2009 (t30).