Project: DRA FRONT END FILTER PROJECT

Title: A HOL Specification of the SWORD Output Filter

Ref: DS/FMU/FEF/024 *Issue: Revision* : *2.1* *Date:* 5 June 2016

Status: Approved *Type:* Specification

Keywords:

Author:

Name	*Location*	*Signature*	*Date*
G. M. Prout	WIN01		

Authorisation for Issue:

Name	*Function*	*Signature*	*Date*
R.B. Jones	HAT Manager		

Abstract: A specification of the SWORD output filter in ProofPower-HOL for the DRA front end filter project RSRE 1C/6130.

Distribution: HAT FEF File
Simon Wiseman

0 DOCUMENT CONTROL

0.1 Contents List

0.2 Document Cross References

[1] DS/FMU/FEF/022. *SWORD Front End Architectural Model.* R.D. Arthan, ICL Secure Systems, WIN01.

[2] DS/FMU/FEF/026. *Critical Requirements on the SWORD Query Transformations.* R.D. Arthan, ICL Secure Systems, WIN01.

[3] *The SWORD Front End Filter Specification.* Simon Wiseman, DRA, 21st January 1993.

0.3 Changes History

Issue 1.1 (4 February 1993) First Draft.

Issue 1.7 (19 May 1993) Added external interface function.

Issue *Revision* : *2.1* **(5 June 2016)** Final approved version.

Issue 2.2 Removed dependency on ICL logo font

0.4 Changes Forecast

Changes may be necessary as a result of issues raised during Phase 2.

1 GENERAL

1.1 Scope

This document gives a formal specification in ProofPower-HOL of the SWORD front end output filter.

1.2 Introduction

In [1] a description of the SWORD architectural model is given. This includes a transition function which has an output filter as one of its subsystems. The output filter removes data that the client is not allowed to see from the output of the TSQL DBMS and decides whether the data may not be complete. This output filter is informally partially specified for select queries in [3]. We give a formal specification in ProofPower-HOL of the output filter for select queries.

2 PRELIMINARIES

The following ProofPower instructions set the context for the proof tools and set up the new theory *fef024*, with parent theory *fef014*, in which the specification of SSQL (and hence TSQL) is given.

```
SML
open_theory"fef022";
(force_delete_theory "fef024" handle _ => ());
new_theory"fef024";
push_pc"hol";
```

3 FILTER FUNCTIONS FOR SELECT QUERIES

We choose to model the filter in two parts. The first part essentially 'undoes' the optimisations carried out in TSQL and the second part filters the data according to the user's clearance.

When an SSQL select query is transformed into a TSQL select query, as well as a table of data, additional information is returned about checks that may be required:

- a boolean is returned which determines whether the class of the where clause has been selected
- an optional class is returned, the constant row existence class, with an indicator if in fact the row existence class has been selected
- a list of booleans is returned which determines whether the field classifications need checking

The first part of the filter operation fills in classification columns that have been optimised away and also returns the class of the where clause and the row existence class for each row. The 'filled in' classes are set at the user's clearance as is the class of the where clause if it has not been selected.

The filter is modelled as if all inputs are correct in the sense that the data lists are in the form as expected by the checks. For example, the length of a data list (after the class of the where clause and the row existence class have been removed) is equal to the number of *false*s plus twice the number of *true*s in the list of booleans that determines whether the field classifications need checking.

3.1 Filling in Selected Rows

A function to retrieve the classification from an item which is to be treated as a classification.

HOL Constant

class_of_item: $Item \rightarrow Class$

$\forall\ i \bullet class\_of\_item\ i = destClassVal(VI\_val(destValuedItem\ i))$

Next, we want to put back classifications that have been optimised away.

The data list is viewed as a value followed by its classification, except in the case where the classification has been optimised away and only the value is present. This is determined by the corresponding list of booleans. The function *fill_cols* takes the user's clearance, the data list and the boolean check list and returns a list of pairs of item and classification with the classification set at the user's clearance if it did not appear in the input list.

HOL Constant

fill_cols: $Class \rightarrow Data\ LIST \times BOOL\ LIST \rightarrow (Item \times Class)\ LIST$

$\forall\ d\ ds\ b\ bs\ clear \bullet$
$fill\_cols\ clear\ ([],[]) = []$
$\wedge\quad fill\_cols\ clear\ (Cons\ d\ ds, Cons\ b\ bs) =$
$\quad if\ \ b\ then$
$\quad\quad let\ c = class\_of\_item(Dat\_item\ (Hd\ ds))$
$\quad\quad in$
$\quad\quad Cons(Dat\_item\ d, c)(fill\_cols\ clear\ (Tl\ ds, bs))$
$\quad else\quad Cons(Dat\_item\ d, clear)(fill\_cols\ clear\ (ds, bs))$

Now we check whether the class of the where clause and the row existence class appear as values in the data list. The function *fill_row* removes these classes from the list if they are there, fills in the column classifications and also returns the class of the where clause and the row existence class. If the class of the where clause does not appear in the data list, then it is returned as the user's clearance.

HOL Constant

fill_row: $BOOL \times (Class + ONE) \times BOOL\ LIST \times Class \rightarrow Data\ LIST \rightarrow Class \times Class \times (Item \times Class)LIST$

$\forall\ check\_where\ check\_rc\ check\_cols\ clear\ dl \bullet$
$fill\_row\ (check\_where,check\_rc,check\_cols,clear)\ dl =$
$\quad let\ (where\_c,rest_1) =$
$\quad\quad if\ check\_where$
$\quad\quad then\ (class\_of\_item(Dat\_item\ (Hd\ dl)),Tl\ dl)$
$\quad\quad else\ (clear,dl)$
$\quad in$
$\quad let\ (rc,rest_2) =$
$\quad if\ IsR\ check\_rc$
$\quad\quad then\ (class\_of\_item(Dat\_item\ (Hd\ rest_1)),Tl\ rest_1)$
$\quad\quad else\ (OutL\ check\_rc,rest_1)$
$\quad in\ (where\_c,rc,fill\_cols\ clear(rest_2,check\_cols))$

Finally, *fill_table* fills in all the missing classifications.

HOL Constant

fill_table: $BOOL \times (Class + ONE) \times BOOL\ LIST \times Class \times Data\ LIST\ LIST \rightarrow (Class \times Class \times (Item \times Class)LIST)LIST$

$\forall\ check\_where\ check\_rc\ check\_cols\ clear\ dll \bullet$
$\quad fill\_table\ (check\_where,check\_rc,check\_cols,clear,dll) =$
$\quad Map\ (fill\_row(check\_where,check\_rc,check\_cols,clear))\ dll$

3.2 Filtering the Selected Data

The filter takes the 'filled' in data and returns the filtered data plus a boolean which is true if the *mayNotBeComplete* message applies.

First check whether the user's clearance dominates the class of the where clause.

HOL Constant

filter_where_row: $Class \times Class \times (Item \times Class)LIST \rightarrow (Item \times Class)LIST \times BOOL$

$\forall\ clear\ where\_c\ icl \bullet$
$filter\_where\_row\ (clear,where\_c,icl) = (icl,\neg(clear\ dominates\ where\_c))$

Throw away any rows about whose existence the user is not cleared to know. In addition, throw away any rows if the where clause is not dominated by the user's clearance. Flag if this has happened so that a *mayNotBeComplete* message can be issued.

HOL Constant

filter_table: $(Class \times Class \times (Item \times Class)LIST)LIST \times Class \rightarrow (Item \times Class)LIST\ LIST \times BOOL$

$\forall\ h\ rest\ clear \bullet$
$filter\_table\ ([],clear) = ([],F)$
$\wedge\quad filter\_table(Cons\ h\ rest,clear) =$
$\quad let\ (where\_c,rc,ics) = h$
$\quad in$
$\quad if\ \neg(clear\ dominates\ rc)$
$\quad then\ filter\_table(rest,clear)$
$\quad else\quad let\ (fics,msg) = filter\_where\_row(clear,where\_c,ics)$
$\quad\quad in$
$\quad\quad let\ (frest,msgs) = filter\_table(rest,clear)$
$\quad\quad in\quad if\ msg\ then\ (frest,T)$
$\quad\quad\quad else\ (Cons\ fics\ frest,msgs)$

Check whether the user is cleared to see the data in a field.

HOL Constant

filter_cols: $Class \rightarrow (Item \times Class)LIST \rightarrow Data\ LIST$

$\forall\ clear\ ic\ ics \bullet$
$filter\_cols\ clear\ [] = []$
$\wedge\quad filter\_cols\ clear\ (Cons\ ic\ ics) =$
$\quad let\ (i,c) = ic$
$\quad in$
$\quad if\ clear\ dominates\ c$
$\quad then\ Cons\ (MkData\ c\ i)(filter\_cols\ clear\ ics)$
$\quad else\quad Cons$
$\quad\quad (MkData\ c(ValuedItemItem(MkValuedItem\ sterling\ dummyVal)))$
$\quad\quad (filter\_cols\ clear\ ics)$

Finally, filter the whole table of selected data. A *mayNotBeComplete* message should be issued if the boolean *true* is returned.

HOL Constant

filter_select: $(Class \times Class \times (Item \times Class)LIST)LIST \times Class \rightarrow Data\ LIST\ LIST \times BOOL$

$\forall\ table\ clear \bullet$
$filter\_select\ (table,clear) =$
  $let\ (ftable,check) = filter\_table(table,clear)$
  in
  $(Map\ (filter\_cols\ clear)\ ftable,check)$

3.3 Interface

We compose the functions of the previous section to give the function *outputFilter* required in [2]. To do this it is convenient to make a type abbreviation for the additional parameter which controls the behaviour of the filter.

SML

```
declare_type_abbrev("FILTER_PARS", [],  ⌜:BOOL × ((Class + ONE) × BOOL LIST)⌝);
```

HOL Constant

outputFilter: $Class \times ANSWER \times FILTER\_PARS \rightarrow ANSWER$

$\forall\ cl\ dl\ errs\ cw\ cr\ cc$
$\bullet$ $outputFilter\ (cl,\ (dl,\ errs),\ cw,\ cr,\ cc)$
$=$ $if\ \neg\ errs = []$
  $then\ ([],\ errs)$
  $else\ let\ (ans,\ mnbc) = filter\_select\ (fill\_table\ (cw,\ cr,\ cc,\ cl,\ dl),\ cl)$
    $in\ (ans,\ if\ mnbc\ then\ [mayNotBeComplete]\ else\ [])$

4 CLOSING DOWN

The following ProofPower instruction restores the previous proof context.

```
SML
|pop_pc();
```

5 INDEX