Project: DRA FRONT END FILTER PROJECT

Title: SWORD Front End Architectural Model

Ref: DS/FMU/FEF/022 *Issue: Revision* : *2.1* *Date:* 5 June 2016

Status: Approved *Type:* Specification

Keywords:

Author:

Name	*Location*	*Signature*	*Date*
R. D. Arthan	WIN01		

Authorisation for Issue:

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

Abstract: The first part of the formal specifications of a model of the SWORD Front End for the DRA front end filter project RSRE 1C/6130.

Distribution: HAT FEF File
Simon Wiseman

0 DOCUMENT CONTROL

0.1 Contents List

0 DOCUMENT CONTROL **2**
0.1 Contents List . . . 2
0.2 Document Cross References . . . 3
0.3 Changes History . . . 3
0.4 Changes Forecast . . . 3

1 GENERAL **4**
1.1 Scope . . . 4
1.2 Introduction . . . 4

2 PRELIMINARIES **4**

3 ARCHITECTURAL MODEL **5**
3.1 Transition Function Construction . . . 5
3.2 Initial State . . . 7
3.3 Behavioural Model . . . 7

4 SUBSYSTEM CORRECTNESS CONJECTURE **7**

5 CLOSING DOWN **12**

6 THE THEORY fef022 **13**
6.1 Parents . . . 13
6.2 Children . . . 13
6.3 Constants . . . 13
6.4 Type Abbreviations . . . 14
6.5 Definitions . . . 14

7 INDEX **17**

0.2 Document Cross References

[1] DS/FMU/017. *Secure Database Technical Proposal*. High Assurance Team, ICL Secure Systems, WIN01, 21st January 1992.

[2] DS/FMU/FEF/003. *Formal Security Policy*. G.M. Prout, ICL Secure Systems, WIN01.

[3] DS/FMU/FEF/005. *Specifications of hide and updateState*. G.M. Prout, ICL Secure Systems, WIN01.

[4] DS/FMU/FEF/006. *Security Conjecture for the SSQL Abstract Machine*. G.M. Prout, ICL Secure Systems, WIN01.

[5] DS/FMU/FEF/007. *Proof Strategy*. G.M. Prout, ICL Secure Systems, WIN01.

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

[7] *The Specification of Secure SQL*. Simon Wiseman, DRA, 6th July 1992.

[8] *SSQL Transformations*. Simon Wiseman, DRA, 14th January 1993.

0.3 Changes History

Issue 1.1 First draft.

Issue 1.6 SSQL Transformation Processor is now allowed to signal errors. The definition of subsystem security conditions is now spread over several boxes.

Issue *Revision* : *2.1* Final approved version.

Issue 2.2 Removed dependency on ICL logo font

0.4 Changes Forecast

None.

1 GENERAL

1.1 Scope

This document gives a formal specification of part of the SWORD Front End giving the high-level security properties required of both the Query Transformations of [8] and of the Front End Filter of [6]. It constitutes part of deliverable D3 of work package 1a, as given in the Phase 2 Technical Proposal, [1].

1.2 Introduction

The Front End implementation of the SWORD database provides an implementation of the secure query language, SSQL, [7], via transformation into the target query language, TSQL, which is a modest extension of the Standard Query Language, SQL. The SWORD Front End thus provides a multi-level secure RDBMS in which both data and security information are stored in a conventional DBMS system implementing TSQL.

This document is intended to serve the following purposes:

- to give a formal model in HOL of the overall architecture of the SWORD Front End, identifying its top-level subsystems and their interconnection;
- to relate the architectural model to the formal specification of the *no flows down* security policy given in [2];
- to identify the critical requirements on the top-level subsystems, which, if met, are conjectured to ensure that the model of the SWORD Front End satisfies the policy.

2 PRELIMINARIES

The following ProofPower instructions set the context for the proof tools and set up the new theory *fef022*, with parent the theory *fef006* in which the SSQL Abstract Machine used to define the semantics of SSQL is formalised.

SML

```
open_theory "fef006";
(force_delete_theory "fef022" handle _ => ());
new_theory"fef022";
push_pc "hol";
```

3 ARCHITECTURAL MODEL

In outline, the SWORD Front End operates as follows. An SSQL query supplied by a client is transformed by the Front End into a TSQL query to perform the operation intended by the SSQL query, together with an optional TSQL query to make certain security checks. The Front End executes the check query, if any, on the conventional DBMS and uses the response to determine whether the SSQL access is permitted (the check query is always a *SELECT* query which returns an empty table if the access is permitted). If the access is permitted, then the other TSQL query is executed to select data from, or to modify, the database. The results of a *SELECT* query are not returned directly to the user, since they may still include data which the user is not cleared to see. Instead, the results are passed through a filter process which overwrites any data that the client may not see and it is the output of the filter which is returned to the client. In some circumstances, the data returned to the client may not be complete, in that there may be rows of data for which it is not permitted for the client to evaluate the SSQL *WHERE* clause; in such cases, an additional advisory message is also returned to the user indicating that the data may not be complete.

The construction of a behavioural model of the SWORD Front End which follows is quite similar to the construction of the SSQL Abstract Machine of [4]: a state-transition function for the system is constructed from various components, an initial state is defined, and then these two are combined to give a behavioural model of the system as a whole. Both the initial state and the behavioural model are parameterised by a function giving the representation of SSQL Abstract Machine states as states of the TSQL DBMS.

3.1 Transition Function Construction

We choose to model the transition function by considering it to be made up from three subsystems as follows:

- The TSQL query processor of the conventional DBMS.
- An SSQL Transformation Processor for performing the syntactic transformation of an SSQL query to give one or two TSQL queries together with any additional parameters required by the other components.
- An Output Filter for removing data which the client is not allowed to see from the output of the TSQL DBMS and for deciding whether the data may not be complete.

The DBMS is modelled using the following types, the first of which is used in the types of the other components:

```
SML
declare_type_abbrev("ANSWER", [],
        ⌜: Data LIST LIST × Errors⌝);

declare_type_abbrev("DBMS_TYPE", ["'TSQL_QUERY", "'ST"],
        ⌜: ('TSQL_QUERY × 'ST) → ('ST × ANSWER)⌝);
```

Note that the data returned by the TSQL DBMS is taken to have the same type as the data returned by the SSQL Abstract Machine of [4]. (This follows [6] and reflects the close relationship between the TSQL and SSQL type systems.)

The remaining components are taken to have types as follows (in which $'TSQL\_QUERY + ONE$ is the disjoint union of $'TSQL\_QUERY$ and a one-point type, i.e., an optional TSQL query).

SML

```
declare_type_abbrev("STP_TYPE", ["'TSQL_QUERY", "'PARS"],
        ⌜: (Query × Class)
                → ('TSQL_QUERY × ('TSQL_QUERY + ONE) × 'PARS) + Errors⌝);

declare_type_abbrev("FILTER_TYPE", ["'PARS"],
        ⌜: (Class × ANSWER × 'PARS) → ANSWER⌝);
```

Note that the type of the SSQL Transformation Processor means that its effect does not depend upon the TSQL state; this reflects the fact that, while it is planned in the SWORD Front End implementation to store information relating to the structure of the database in the TSQL DBMS, in the model, it is assumed that the structure of the database is fixed.

The transition function itself has the following type:

SML

```
declare_type_abbrev("TF_TYPE", ["'ST"],
        ⌜: ((Query × Class) × 'ST) → ('ST × (Class × ANSWER))⌝);
```

The construction of the transition function from the four subsystems is as follows:

HOL Constant

```
        mkTf_f: ('TSQL_QUERY, 'ST) DBMS_TYPE
                        → ('TSQL_QUERY, 'PARS) STP_TYPE
                        → 'PARS FILTER_TYPE
                        → 'ST TF_TYPE
├──────────────────────────────────────────────────────────
∀ dbms stp filter q c st•
        mkTf_f dbms stp filter ((q, c), st)
=       let     res = stp(q, c)
        in      if isError res
                then    (st, (c, [], destError res))
                else    let     (dq, ocq, pars) = destVal res
                        in let  (st', (cks, errs)) =
                                        if IsL ocq
                                        then    dbms (OutL ocq, st)
                                        else    (st, ([], []))
                        in      if cks = [] ∧ errs = []
                                then    let     (st'', ans2) = dbms (dq, st')
                                        in      (st'', (c, filter (c, ans2, pars)))
                                else    (st', (c, [], [notCleared]))
```

3.2 Initial State

In order to specify the initial state, and later to define the security requirements on the subsystems, we need to have available a function to map any state of the SSQL abstract machine of [4] to the TSQL state which represents it. The initial TSQL state is the result of representing the initial SSQL state.

HOL Constant

$$\mathbf{istate_f} : (State \to 'ST) \to 'ST$$

$$\forall repr \bullet istate_f\ repr = repr\ isstate$$

(The initial state is presented in this way, rather than just by using *repr isstate* in the behavioural model, to emphasise the analogy between the construction of the Front End Implementation model and the SSQL abstract machine.)

3.3 Behavioural Model

The top-level structure of the Front End implementation of SWORD is then captured in the following definition, which defines how subsystems of the appropriate types are combined to make a behavioural model of the system in the sense of the formal security policy of [2]. Like the initial state, the model is parameterised by the function representing SSQL states as TSQL states in addition to the subsystems proper.

HOL Constant

$$\begin{aligned}&\mathbf{FE\_SWORD}: (State \to 'ST)\\ &\qquad \to ('TSQL\_QUERY, 'ST)\ DBMS\_TYPE\\ &\qquad \to ('TSQL\_QUERY, 'PARS)\ STP\_TYPE\\ &\qquad \to 'PARS\ FILTER\_TYPE\\ &\qquad \to (Query, ANSWER)\ BEHAVIOURS\end{aligned}$$

$$\begin{aligned}&\forall repr\ dbms\ stp\ filter \bullet\\ &\qquad FE\_SWORD\ repr\ dbms\ stp\ filter\\ =\ &\qquad behaviours\ (mkTf_f\ dbms\ stp\ filter, istate_f\ repr)\end{aligned}$$

4 SUBSYSTEM CORRECTNESS CONJECTURE

For a given representation function, *repr*, and subsystems, *dbms*, *stp*, *filter*, and *check*, the security policy of [2] applies directly:

$$?\vdash \quad FE\_SWORD\ repr\ dbms\ stp\ filter\ check \in secure$$

But, of course, this places stringent requirements on the parameters to *FE_SWORD*. In order to formulate these requirements, we use the *hide* function of [3] and quantification over the SSQL states

to make the necessary assertions about information flow in the TSQL states. Of necessity, at the level of abstraction taken so far in this document, it is only possible to make these assertions as properties of the two or more of the three subsystems taken together.

The following formulation has been derived by attempting to give necessary conditions on the results of the SSQL Transformation Processor to ensure that a secure system results for a given DBMS and filter operation. It is largely based on the analogous decomposition of the security conjecture in [5]. As with the initial state and behavioural model, the conditions are parameterised as necessary by the represention function, *repr*.

The following function determines whether the checks to be imposed before issuing the data query are satisfied. Its first parameter is the TSQL DBMS. The second parameter is the result returned by the SSQL Transformation Processor. If this parameter is an error indicator then there is no data query to issue. Otherwise, if the parameter does contain the optional check query then the data query is only to be issued if execution of the check query in the state given by the third parameter produces no errors and no rows of output

HOL Constant

$$
\begin{array}{l}
\quad \textbf{ok\_to\_proceed} \quad : \ (('TSQL\_QUERY, 'ST)\ DBMS\_TYPE) \\
\qquad\qquad\qquad \rightarrow (('TSQL\_QUERY \times ('TSQL\_QUERY + ONE) \times 'PARS) + Errors\ ) \\
\qquad\qquad\qquad \rightarrow 'ST \\
\qquad\qquad\qquad \rightarrow BOOL \\
\hline
\forall dbms\ st\ stp\_res \bullet \\
\quad ok\_to\_proceed\ dbms\ stp\_res\ st \\
\Leftrightarrow \quad \neg isError\ stp\_res \\
\quad \wedge \quad let \quad (dq,\ ocq,\ pars) = destVal\ stp\_res \\
\qquad in \quad IsL\ ocq \\
\qquad \Rightarrow \quad let \quad (cks,\ errs) = Snd(dbms(OutL\ ocq,\ st)) \\
\qquad\quad in \quad errs = [] \wedge cks = []
\end{array}
$$

Informally, what we want to say is that the components *dbms*, *stp* and *filter* are secure subsystems with respect to a given representation function *repr* if the data query, *dq*, optional check query, *ocq* and parameters, *pars* returned by *stp* for any query, *q*, at any clearance, *c*, have the following properties (where we say that "the check fails for a state" if the transformation processing reported an error, or there is an optional check query, *ocq*, and its execution either gives or returns one or more rows of output).

We now give five conditions which we assert will guarantee the desired overall security property. Each condition is defined as a set. The first four conditions do not depend on the *filter* component and so are defined as sets of DBMS-SSQL Transformation Processor pairs (parameterised by the representation function). The fifth condition involves the filter as well and so is a set of triples (again parameterised by the representation function). The conditions follow:

(*A*) Execution of *dq* on any state for which the check succeeds preserves all the structural constraints which allow a TSQL state to be viewed in a unique way as an SSQL state:

HOL Constant

$$\begin{array}{l} \quad \textbf{subsys\_secureA} \quad : (State \to 'ST) \\ \qquad\qquad\qquad\quad \to (('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \\ \qquad\qquad\qquad\quad \times\ ('TSQL\_QUERY, 'PARS)\ STP\_TYPE)\ SET \\ \hline \\ \forall repr\ dbms\ stp \bullet \\ \quad (dbms,\ stp) \in subsys\_secureA\ repr \\ \Leftrightarrow \quad \forall q\ c \bullet \\ \quad let \quad res = stp(q,\ c) \\ \quad in\ let \quad (dq,\ ocq,\ pars) = destVal\ (stp(q,\ c)) \\ \quad in\ let \quad ok\ s = ok\_to\_proceed\ dbms\ res\ s \\ \quad in \quad \forall s \bullet \exists_1 s' \bullet\ ok\ (repr\ s) \Rightarrow repr\ s' = Fst(dbms(dq,\ repr\ s)) \end{array}$$

(B) If there is a query in *ocq* then its execution does not change the state (i.e., it is a *SELECT* query) and gives rise to no errors:

HOL Constant

$$\begin{array}{l} \quad \textbf{subsys\_secureB} \quad : (State \to 'ST) \\ \qquad\qquad\qquad\quad \to (('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \\ \qquad\qquad\qquad\quad \times\ ('TSQL\_QUERY, 'PARS)\ STP\_TYPE)\ SET \\ \hline \\ \forall repr\ dbms\ stp \bullet \\ \quad (dbms,\ stp) \in subsys\_secureB\ repr \\ \Leftrightarrow \quad \forall q\ c \bullet \\ \quad let \quad res = stp(q,\ c) \\ \quad in\ let \quad (dq,\ ocq,\ pars) = destVal\ (stp(q,\ c)) \\ \quad in \quad \forall st_1 \bullet\ \neg isError\ res \wedge IsL\ ocq \\ \qquad \Rightarrow \quad let \quad (st_2,\ (cks,\ errs)) = dbms(OutL\ ocq,\ st_1) \\ \qquad\quad in \quad st_2 = st_1 \wedge errs = [] \end{array}$$

(C) If for some clearance c', execution of dq changes the state in a way which is visible at clearance c', and c' does not dominate c, then the check must fail:

HOL Constant

$$\begin{array}{l}
\quad \textbf{subsys_secureC} \quad : (State \to 'ST) \\
\qquad\qquad\qquad\quad \to (('TSQL_QUERY, 'ST)\ DBMS_TYPE \\
\qquad\qquad\qquad\quad \times\ ('TSQL_QUERY, 'PARS)\ STP_TYPE)\ SET \\
\hline
\forall repr\ dbms\ stp \bullet \\
\quad (dbms,\ stp) \in subsys_secureC\ repr \\
\Leftrightarrow \quad \forall q\ c \bullet \\
\quad let \quad res = stp(q,\ c) \\
\quad in\ let \quad (dq,\ ocq,\ pars) = destVal\ (stp(q,\ c)) \\
\quad in\ let \quad ok\ s = ok_to_proceed\ dbms\ res\ s \\
\quad in \quad (\forall c'\ s'\ s\ d \bullet \\
\qquad\qquad (repr\ s',\ d) = dbms(dq,\ repr\ s) \\
\qquad\quad \wedge \quad \neg hide(c',\ s) = hide(c',\ s') \\
\qquad\quad \wedge \quad \neg c'\ dominates\ c \\
\qquad\quad \Rightarrow \quad \neg ok(repr\ s))
\end{array}$$

(D) If the states resulting from execution of dq reveal a distinction which a client at clearance c', dominating c, is not allowed to see then the check must fail:

HOL Constant

$$\begin{array}{l}
\quad \textbf{subsys_secureD} \quad : (State \to 'ST) \\
\qquad\qquad\qquad\quad \to (('TSQL_QUERY, 'ST)\ DBMS_TYPE \\
\qquad\qquad\qquad\quad \times\ ('TSQL_QUERY, 'PARS)\ STP_TYPE)\ SET \\
\hline
\forall repr\ dbms\ stp \bullet \\
\quad (dbms,\ stp) \in subsys_secureD\ repr \\
\Leftrightarrow \quad \forall q\ c \bullet \\
\quad let \quad res = stp(q,\ c) \\
\quad in\ let \quad (dq,\ ocq,\ pars) = destVal\ (stp(q,\ c)) \\
\quad in\ let \quad ok\ s = ok_to_proceed\ dbms\ res\ s \\
\quad in \quad (\forall c'\ s_1\ s_2\ s'_1\ s'_2\ d_1\ d_2 \bullet \\
\qquad\qquad hide(c,\ s_1) = hide(c,\ s_2) \\
\qquad\quad \wedge \quad (repr\ s'_1,\ d_1) = dbms(dq,\ repr\ s_1) \\
\qquad\quad \wedge \quad (repr\ s'_2,\ d_2) = dbms(dq,\ repr\ s_2) \\
\qquad\quad \wedge \quad c'\ dominates\ c \\
\qquad\quad \wedge \quad \neg hide(c',\ s'_1) = hide(c',\ s'_2) \\
\qquad\quad \Rightarrow \quad \neg ok(repr\ s_1))
\end{array}$$

(E) If the outputs resulting from execution of dq reveal a distinction which a client at clearance c is not allowed to see between two states, then the check must fail:

HOL Constant

$$\begin{array}{l}
\quad \textbf{subsys_secureE} \quad : (State \to 'ST) \\
\qquad\qquad \to (('TSQL_QUERY, 'ST)\ DBMS_TYPE \\
\qquad\qquad \times\ ('TSQL_QUERY, 'PARS)\ STP_TYPE \\
\qquad\qquad \times\ 'PARS\ FILTER_TYPE)\ SET \\
\hline
\forall repr\ dbms\ stp\ filter\bullet \\
\quad (dbms,\ stp,\ filter) \in subsys_secureE\ repr \\
\Leftrightarrow \quad \forall q\ c\bullet \\
\quad let \quad res = stp(q,\ c) \\
\quad in\ let \quad (dq,\ ocq,\ pars) = destVal\ (stp(q,\ c)) \\
\quad in\ let \quad ok\ s = ok_to_proceed\ dbms\ res\ s \\
\quad in \quad (\forall s_1\ s_2\ s'_1\ s'_2\ d_1\ d_2\bullet \\
\qquad\qquad hide(c,\ s_1) = hide(c,\ s_2) \\
\qquad\quad \wedge \quad (repr\ s'_1,\ d_1) = dbms(dq,\ repr\ s_1) \\
\qquad\quad \wedge \quad (repr\ s'_2,\ d_2) = dbms(dq,\ repr\ s_2) \\
\qquad\quad \wedge \quad \neg filter(c,\ d_1,\ pars) = filter(c,\ d_2,\ pars) \\
\qquad\quad \Rightarrow \quad \neg ok(repr\ s_1))
\end{array}$$

Remarks Clearly condition *A* makes little sense unless *repr* is 1-1. The space optimisations made in the transformations imply that one cannot recover the SSQL state from the TSQL state alone, since, for example, without knowledge of the structure of the SSQL table it represents, one cannot tell whether the first column of a TSQL table contains row existence classes, or column classes or data (according as whether the row class, first column class or both are known to be constant). It is assumed, therefore, that both the transformations and the representation function will be defined using some loosely specified, but constant, specification of the structure of the SSQL state (essentially, the structure implicit in the loosely defined initial abstract state, *isstate*, of [4]).

Clauses *C*, *D* and *E* correspond to clauses *1*, *2* and *3* of the *4* clauses of the relation *secureUpdate* of [5], which captures the security properties on the relationship between the *hide* and *updateState* components of the SSQL Abstract Machine. Clause *4* of *secureUpdate* just says that the client clearance is faithfully passed on from the input to the output, and is ensured by the definition of $mkTf_t$.

We now put the five conditions together:

HOL Constant

$$\begin{array}{l} \textbf{subsys\_secure} \quad : (State \to 'ST) \\ \qquad \to (('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \\ \qquad \times\ ('TSQL\_QUERY, 'PARS)\ STP\_TYPE \\ \qquad \times\ 'PARS\ FILTER\_TYPE)\ SET \end{array}$$

$$\begin{array}{lll} \forall repr\ dbms\ stp\ filter \bullet & & \\ \quad (dbms, stp, filter) \in subsys\_secure\ repr & & \\ \Leftrightarrow & (dbms, stp) & \\ \quad \in & subsys\_secureA\ repr & \\ & \cap & subsys\_secureB\ repr \\ & \cap & subsys\_secureC\ repr \\ & \cap & subsys\_secureD\ repr \\ \quad \wedge & (dbms, stp, filter) & \\ \quad \in & subsys\_secureE\ repr & \end{array}$$

The correctness conjecture for the architectural model presented in this document is then the following conjecture

$$\begin{array}{lll} ?\vdash & \forall repr\ dbms\ stp\ filter \bullet & \\ & (dbms, stp, filter) \in subsys\_secure\ repr & \\ & \Rightarrow & FE\_SWORD\ repr\ dbms\ stp\ filter \in secure \end{array}$$

5 CLOSING DOWN

The following ProofPower instruction restores the previous proof context.

SML

```
pop_pc();
```

6 THE THEORY fef022

6.1 Parents

fef006

6.2 Children

fef024

6.3 Constants

mkTf$_f$ $('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \to ('TSQL\_QUERY, 'PARS)\ STP\_TYPE \to 'PARS\ FILTER\_TYPE \to 'ST\ TF\_TYPE$

istate$_f$ $(State \to 'ST) \to 'ST$

FE_SWORD $(State \to 'ST) \to ('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \to ('TSQL\_QUERY, 'PARS)\ STP\_TYPE \to 'PARS\ FILTER\_TYPE \to (Query, ANSWER)\ BEHAVIOURS$

ok_to_proceed $('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \to ('TSQL\_QUERY \times 'TSQL\_QUERY + Maybe \times 'PARS) + Errors \to 'ST \to Bool$

subsys_secureA $(State \to 'ST) \to ('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \leftrightarrow ('TSQL\_QUERY, 'PARS)\ STP\_TYPE$

subsys_secureB $(State \to 'ST) \to ('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \leftrightarrow ('TSQL\_QUERY, 'PARS)\ STP\_TYPE$

subsys_secureC $(State \to 'ST) \to ('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \leftrightarrow ('TSQL\_QUERY, 'PARS)\ STP\_TYPE$

subsys_secureD $(State \to 'ST) \to ('TSQL\_QUERY, 'ST)\ DBMS\_TYPE \leftrightarrow ('TSQL\_QUERY, 'PARS)\ STP\_TYPE$

subsys_secureE

$(State \to 'ST)$
$\to ('TSQL\_QUERY, 'ST)\ DBMS\_TYPE$
$\leftrightarrow (('TSQL\_QUERY, 'PARS)\ STP\_TYPE$
$\times 'PARS\ FILTER\_TYPE)$

subsys_secure

$(State \to 'ST)$
$\to ('TSQL\_QUERY, 'ST)\ DBMS\_TYPE$
$\leftrightarrow (('TSQL\_QUERY, 'PARS)\ STP\_TYPE$
$\times 'PARS\ FILTER\_TYPE)$

6.4 Type Abbreviations

ANSWER $ANSWER$

$('TSQL\_QUERY, 'ST)$ **DBMS_TYPE**
$('TSQL\_QUERY, 'ST)\ DBMS\_TYPE$

$('TSQL\_QUERY, 'PARS)$ **STP_TYPE**
$('TSQL\_QUERY, 'PARS)\ STP\_TYPE$

$'PARS$ **FILTER_TYPE**
$'PARS\ FILTER\_TYPE$

$'ST$ **TF_TYPE** $'ST\ TF\_TYPE$

6.5 Definitions

mkTf$_f$
$\vdash \forall\ dbms\ stp\ filter\ q\ c\ st$
$\bullet\ mkTf_f\ dbms\ stp\ filter\ ((q,\ c),\ st)$
$= (let\ res = stp\ (q,\ c)$
$in\ if\ isError\ res$
$then\ (st,\ c,\ [],\ destError\ res)$
$else$
$(let\ (dq,\ ocq,\ pars) = destVal\ res$
$in\ let\ (st',\ cks,\ errs)$
$= (if\ IsL\ ocq$
$then\ dbms\ (OutL\ ocq,\ st)$
$else\ (st,\ [],\ []))$
$in\ if\ cks = [] \wedge errs = []$
$then$
$let\ (st'',\ ans2) = dbms\ (dq,\ st')$
$in\ (st'',\ c,\ filter\ (c,\ ans2,\ pars))$
$else\ (st',\ c,\ [],\ [notCleared])))$

istate$_f$
$\vdash \forall\ repr \bullet\ istate_f\ repr = repr\ isstate$

FE_SWORD
$\vdash \forall\ repr\ dbms\ stp\ filter$
$\bullet\ FE\_SWORD\ repr\ dbms\ stp\ filter$
$= behaviours$
$(mkTf_f\ dbms\ stp\ filter,\ istate_f\ repr)$

ok_to_proceed
$\vdash \forall\ dbms\ st\ stp\_res$
$\bullet\ ok\_to\_proceed\ dbms\ stp\_res\ st$
$\Leftrightarrow \neg\ isError\ stp\_res$

$$\wedge\ (let\ (dq,\ ocq,\ pars) = destVal\ stp\_res\ in\ IsL\ ocq \Rightarrow (let\ (cks,\ errs) = Snd\ (dbms\ (OutL\ ocq,\ st))\ in\ errs = [] \wedge cks = []))$$

subsys_secureA

$$\vdash \forall\ repr\ dbms\ stp \bullet (dbms,\ stp) \in subsys\_secureA\ repr \Leftrightarrow (\forall\ q\ c \bullet (let\ res = stp\ (q,\ c)\ in\ let\ (dq,\ ocq,\ pars) = destVal\ (stp\ (q,\ c))\ in\ let\ ok\ s \Leftrightarrow ok\_to\_proceed\ dbms\ res\ s\ in\ \forall\ s \bullet \exists_1\ s' \bullet ok\ (repr\ s) \Rightarrow repr\ s' = Fst\ (dbms\ (dq,\ repr\ s))))$$

subsys_secureB

$$\vdash \forall\ repr\ dbms\ stp \bullet (dbms,\ stp) \in subsys\_secureB\ repr \Leftrightarrow (\forall\ q\ c \bullet (let\ res = stp\ (q,\ c)\ in\ let\ (dq,\ ocq,\ pars) = destVal\ (stp\ (q,\ c))\ in\ \forall\ st_1 \bullet \neg\ isError\ res \wedge IsL\ ocq \Rightarrow (let\ (st_2,\ cks,\ errs) = dbms\ (OutL\ ocq,\ st_1)\ in\ st_2 = st_1 \wedge errs = [])))$$

subsys_secureC

$$\vdash \forall\ repr\ dbms\ stp \bullet (dbms,\ stp) \in subsys\_secureC\ repr \Leftrightarrow (\forall\ q\ c \bullet (let\ res = stp\ (q,\ c)\ in\ let\ (dq,\ ocq,\ pars) = destVal\ (stp\ (q,\ c))\ in\ let\ ok\ s \Leftrightarrow ok\_to\_proceed\ dbms\ res\ s\ in\ \forall\ c'\ s'\ s\ d \bullet (repr\ s',\ d) = dbms\ (dq,\ repr\ s) \wedge \neg\ hide\ (c',\ s) = hide\ (c',\ s') \wedge \neg\ c'\ dominates\ c \Rightarrow \neg\ ok\ (repr\ s)))$$

subsys_secureD

$$\vdash \forall\ repr\ dbms\ stp \bullet (dbms,\ stp) \in subsys\_secureD\ repr \Leftrightarrow (\forall\ q\ c \bullet (let\ res = stp\ (q,\ c)\ in\ let\ (dq,\ ocq,\ pars) = destVal\ (stp\ (q,\ c))\ in\ let\ ok\ s \Leftrightarrow ok\_to\_proceed\ dbms\ res\ s\ in\ \forall\ c'\ s_1\ s_2\ s'_1\ s'_2\ d_1\ d_2$$

$$
\begin{array}{l}
\bullet\ hide\ (c,\ s_1) = hide\ (c,\ s_2) \\
\quad \wedge\ (repr\ s'_1,\ d_1) \\
\quad\quad = dbms\ (dq,\ repr\ s_1) \\
\quad \wedge\ (repr\ s'_2,\ d_2) \\
\quad\quad = dbms\ (dq,\ repr\ s_2) \\
\quad \wedge\ c'\ dominates\ c \\
\quad \wedge\ \neg\ hide\ (c',\ s'_1) \\
\quad\quad\ = hide\ (c',\ s'_2) \\
\Rightarrow\ \neg\ ok\ (repr\ s_1)))
\end{array}
$$

subsys_secureE

$$
\begin{array}{l}
\vdash\ \forall\ repr\ dbms\ stp\ filter \\
\bullet\ (dbms,\ stp,\ filter) \in subsys\_secureE\ repr \\
\Leftrightarrow\ (\forall\ q\ c \\
\bullet\ (let\ res = stp\ (q,\ c) \\
in\ let\ (dq,\ ocq,\ pars) = destVal\ (stp\ (q,\ c)) \\
in\ let\ ok\ s \Leftrightarrow ok\_to\_proceed\ dbms\ res\ s \\
in\ \forall\ s_1\ s_2\ s'_1\ s'_2\ d_1\ d_2 \\
\bullet\ hide\ (c,\ s_1) = hide\ (c,\ s_2) \\
\quad \wedge\ (repr\ s'_1,\ d_1) \\
\quad\quad = dbms\ (dq,\ repr\ s_1) \\
\quad \wedge\ (repr\ s'_2,\ d_2) \\
\quad\quad = dbms\ (dq,\ repr\ s_2) \\
\quad \wedge\ \neg\ filter\ (c,\ d_1,\ pars) \\
\quad\quad\ = filter\ (c,\ d_2,\ pars) \\
\Rightarrow\ \neg\ ok\ (repr\ s_1)))
\end{array}
$$

subsys_secure

$$
\begin{array}{l}
\vdash\ \forall\ repr\ dbms\ stp\ filter \\
\bullet\ (dbms,\ stp,\ filter) \in subsys\_secure\ repr \\
\Leftrightarrow\ (dbms,\ stp) \\
\in\ subsys\_secureA\ repr \\
\cap\ subsys\_secureB\ repr \\
\cap\ subsys\_secureC\ repr \\
\cap\ subsys\_secureD\ repr \\
\wedge\ (dbms,\ stp,\ filter) \in subsys\_secureE\ repr
\end{array}
$$

7 INDEX

ANSWER 5
DBMS_TYPE 5
fef022 4
FE_SWORD 7
FILTER_TYPE 6
$istate_f$ 7
$mkTf_f$ 6
ok_to_proceed 8
STP_TYPE 6
subsys_secureA 9
subsys_secureB 9
subsys_secureC 10
subsys_secureD 10
subsys_secureE 11
subsys_secure 12
TF_TYPE 6