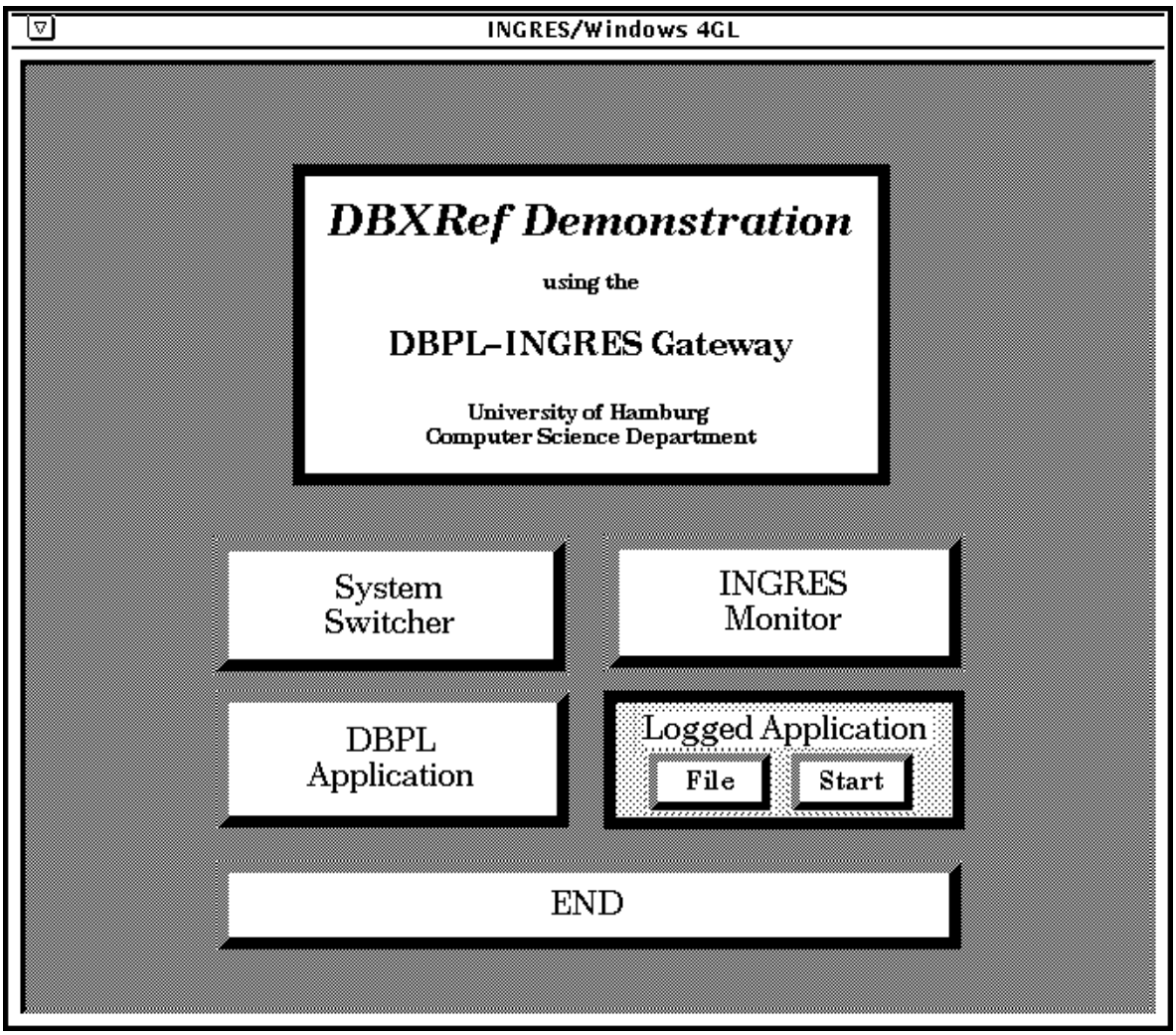
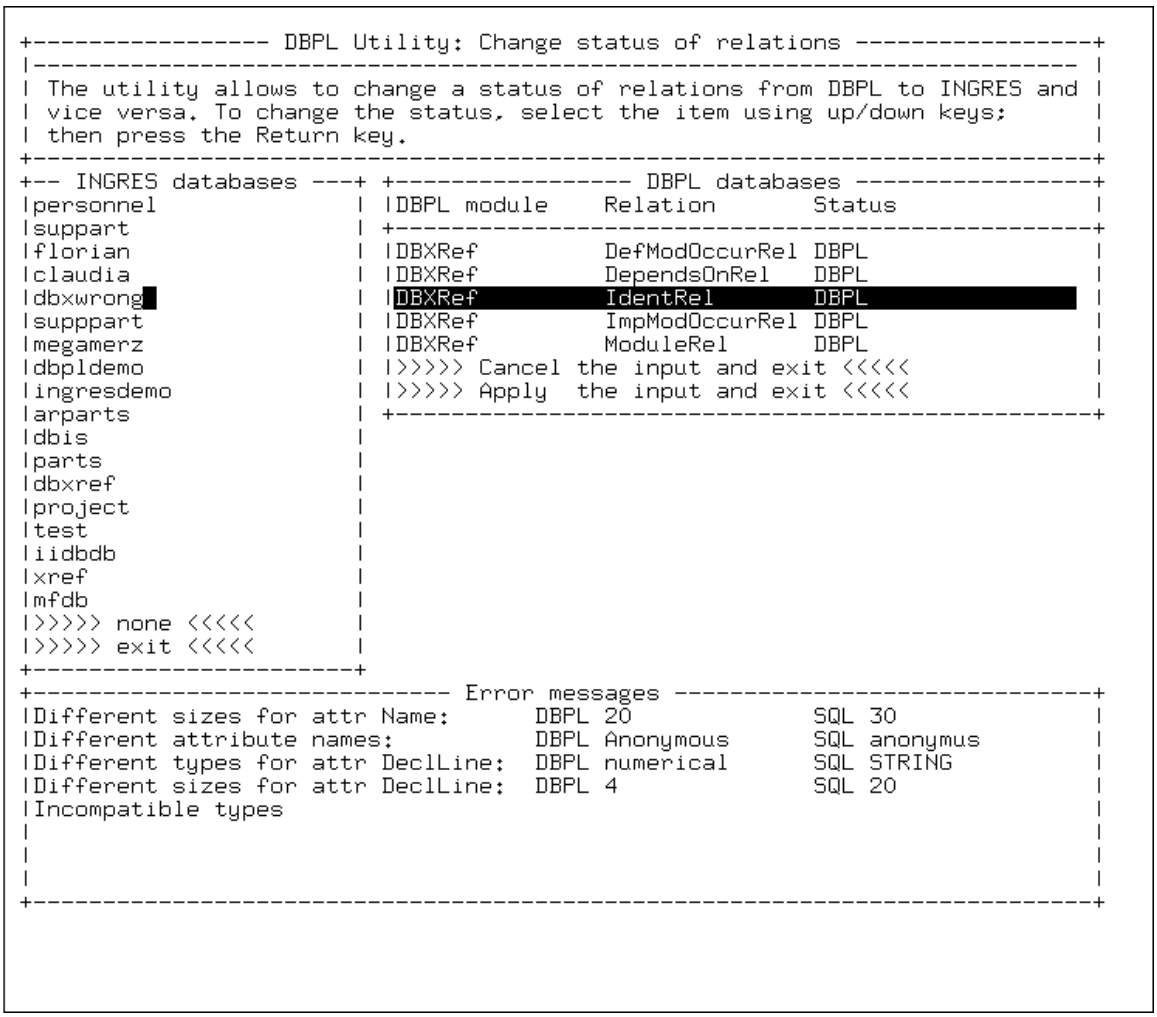
Hallo Florian,
dies ist der Pfad durch die DBPL Demo, den ich versucht habe, mit Hilfe Deiner Notizen zu rekonstruieren

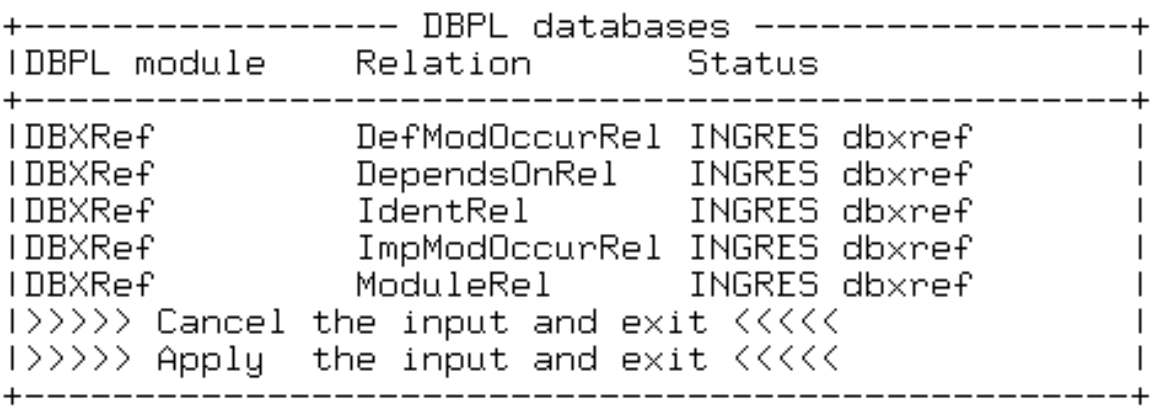


Zuerst erfolgt der Bindungsversuch mit der nichtgeeigneten INGRES DB *dbxwrong*:

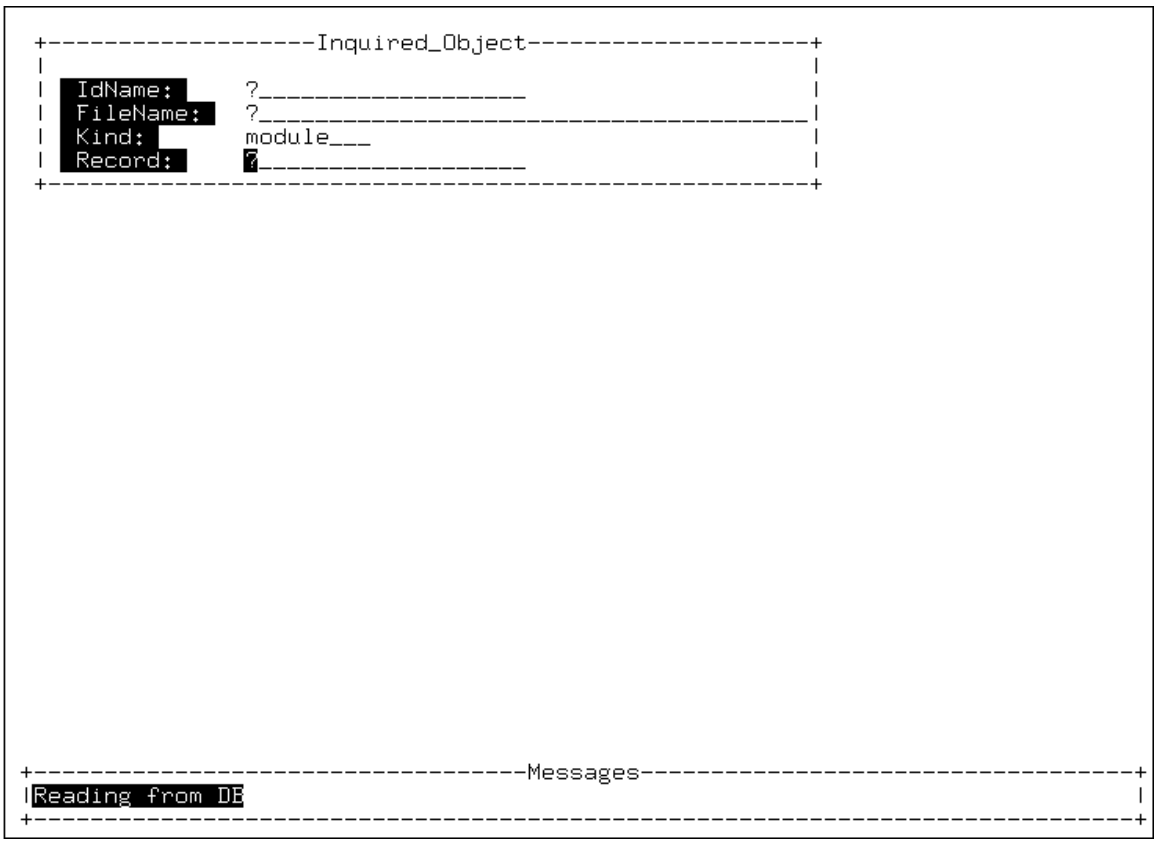
- `DefModOccurRel` fehlt das Attribut `line`
- `DependsOnRel` gibt es nicht
- `IdentRel` hat inkompatible Attributtypen
- die restlichen beiden Relationen sind geeignet



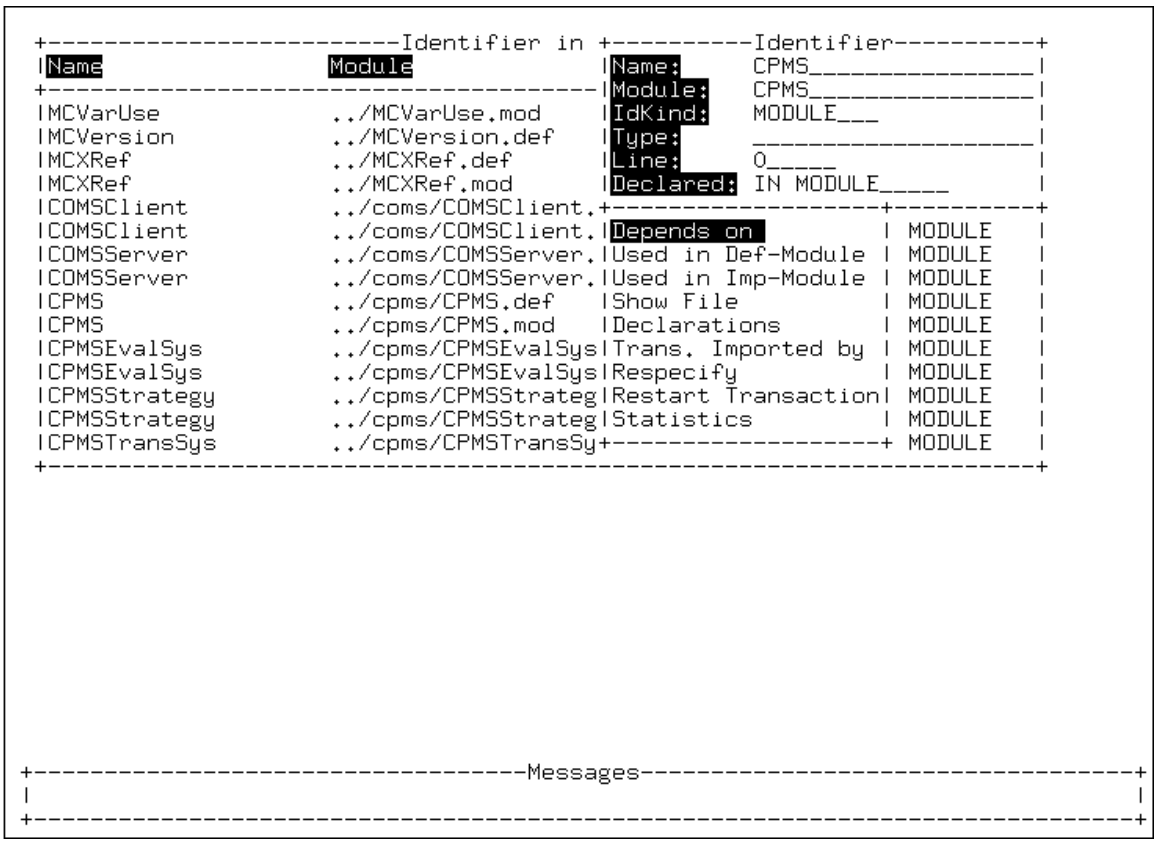
Dann erfolgt die Bindung an die geeignete INGRES DB *dbxref*



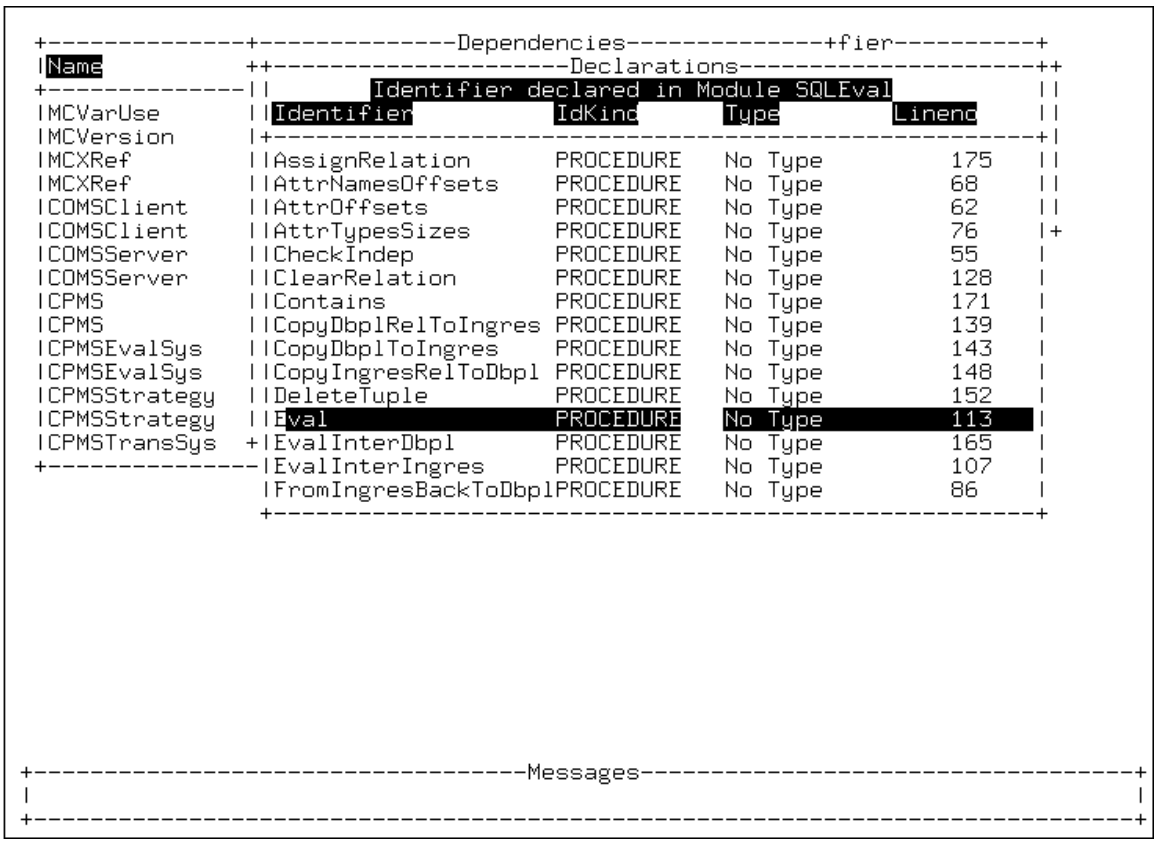
Nach der Selektion aller Module



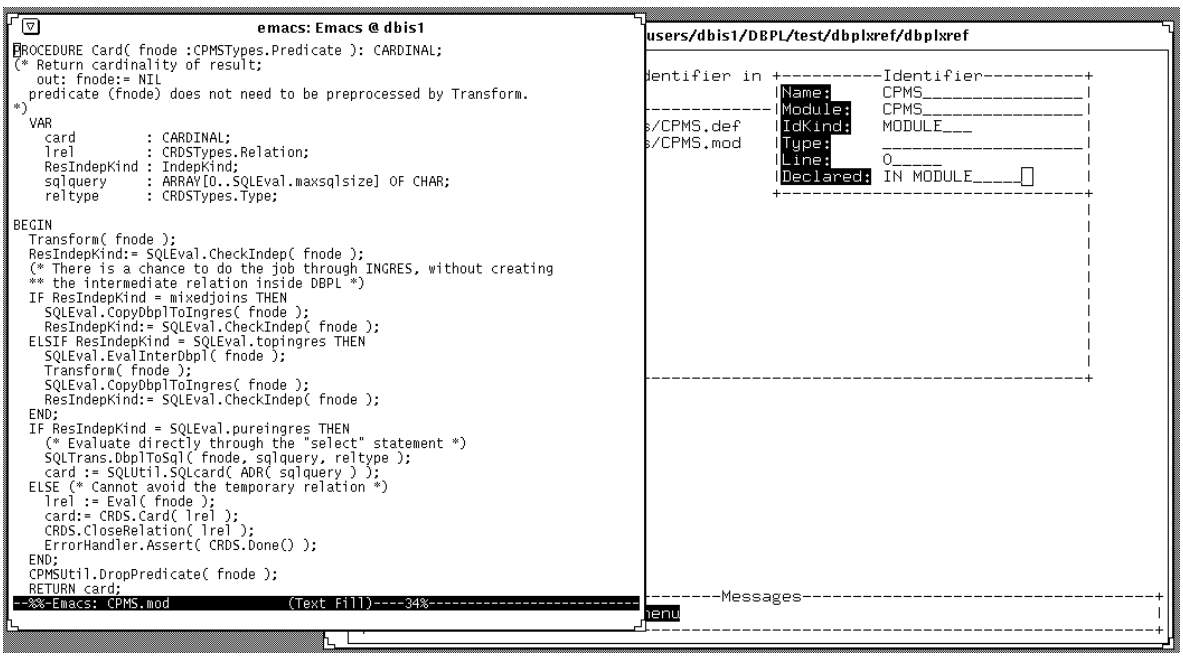
wird das Modul `CPMS.mod` ausgewählt, wobei dafür die *depends on* Module angezeigt werden



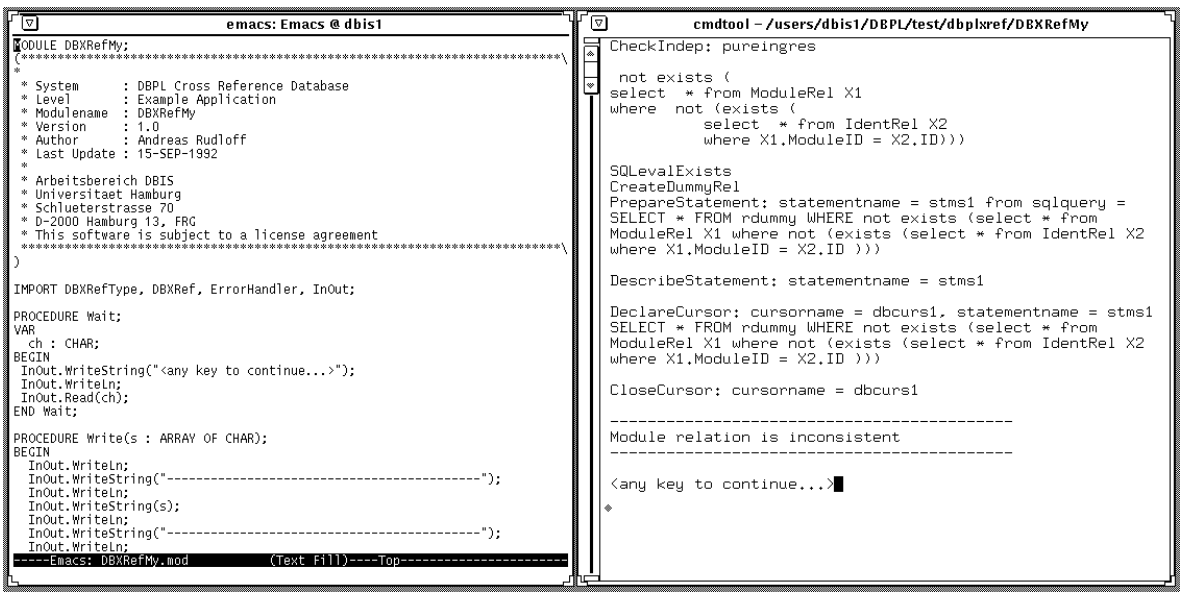
Dabei sind besonders `SQLEval` und `SQLTrans` interessant sind. So können dann für `SQLEval` alle *Declarations* angezeigt werden.



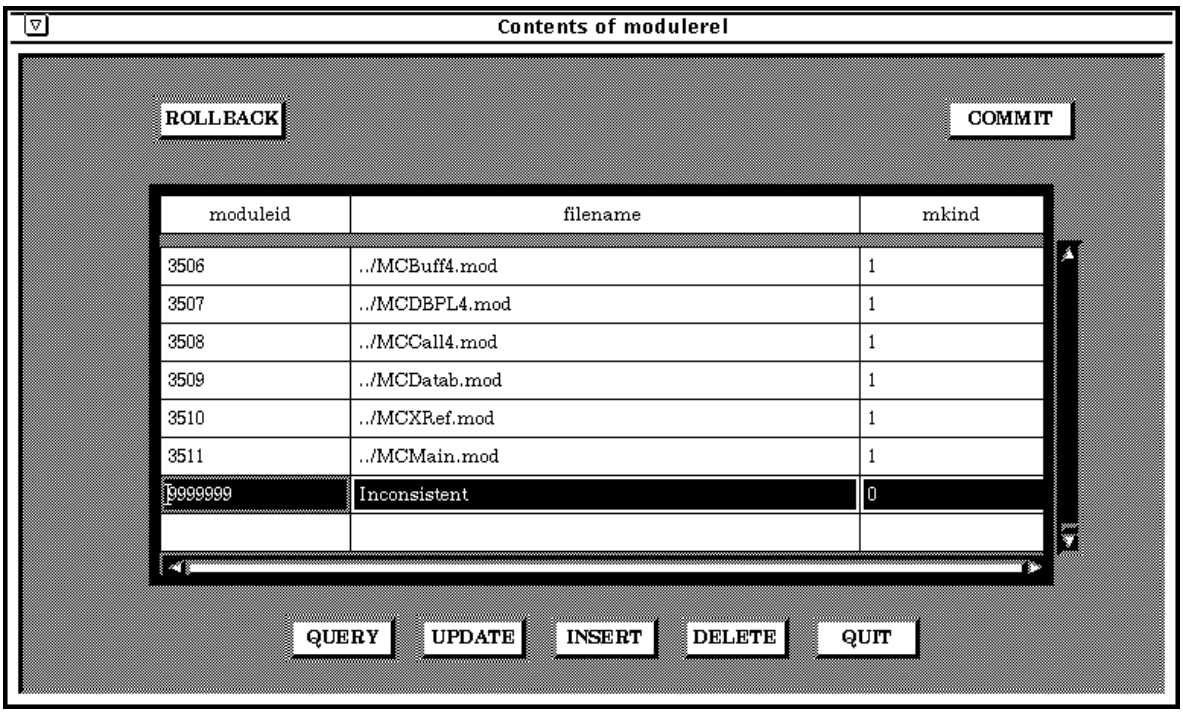
Eine darin interessante Prozedur (z.B. `Eval`) kann dann weiterverfolgt und mit *Show File* in den Emacs geladen werden (alternativ vielleicht `SQLTrans` mit `DbplToSql`; es gibt auch einen Snapshot mit den SQLTrans-deklarationen
Nach dem Zurücknavigieren zu `CPMS` wird der Quelltext geladen und `PROCEDURE CARD` gesucht.



Nun sollte ein `Respecify` durchgeführt werden, damit DBPL alle Sperren freigibt (da es inbes. Schreibsperren hält. Dann kann `DBXRefMy` (dies ist die *Logged Application* im Editor kurz gezeigt werden (*File*), worauf der *Start* erfolgen kann.



Die Ausführung bleibt am Ende der Konsistenzueberpruefung (vor EOT) stehen, damit die Ausgabe erklaert werden kann. Nach `RETURN` wird nun ein Tupel in die `ModuleRel` eingefügt und wieder die Konsistenzprüfung mit dem Stop vor EOT durchgeführt. Nun kann der INGRES Monitor gestartet werden. Ein *Inquire* auf die `ModuleRel` gefolgt von `Query` zeigt alle Module an, wobei das durch `DBXRefMy` eingefügte in der letzten Zeile steht. Dies wird nun ausgewählt und versucht zu löschen. Die Löschung findet aber erst nach dem `RETURN` in der laufenden Anwendung statt. Nun kann `DBXRefMy` nochmal gestartet werden, die Anwendung wird jedoch erst beginnen, wenn nun seinerseits im INGRES Monitor *commit* gewählt wurde, wobei dann auch die Relation wieder konsistent ist.



Die statistischen Daten sind "ubrigens:

- 3509 Identifiers
- 223 Modules
- 5135 Dependencies
- 3358 Occurences Def.
- 40596 Occurences Imp.

Die o.g Programmstücke direkt in Latex (eventuell sind sie so besser lesbar):
`CPMS.Card:`

```
PROCEDURE Card( fnode :CPMSTypes.Predicate ): CARDINAL;
(* Return cardinality of result;
   out: fnode:= NIL
  predicate (fnode) does not need to be preprocessed by Transform.
*)
  VAR
    card         : CARDINAL;
    lrel         : CRDSTypes.Relation;
    ResIndepKind : IndepKind;
    sqlquery     : ARRAY[0..SQLEval.maxsqlsize] OF CHAR;
    reltype      : CRDSTypes.Type;

BEGIN
  Transform( fnode );
```

```
  ResIndepKind:= SQLEval.CheckIndep( fnode );
  (* There is a chance to do the job through INGRES, without creating
  ** the intermediate relation inside DBPL *)
  IF ResIndepKind = mixedjoins THEN
    SQLEval.CopyDbplToIngres( fnode );
    ResIndepKind:= SQLEval.CheckIndep( fnode );
  ELSIF ResIndepKind = SQLEval.topingres THEN
    SQLEval.EvalInterDbpl( fnode );
    Transform( fnode );
    SQLEval.CopyDbplToIngres( fnode );
    ResIndepKind:= SQLEval.CheckIndep( fnode );
  END;
  IF ResIndepKind = SQLEval.pureingres THEN
    (* Evaluate directly through the "select" statement *)
    SQLTrans.DbplToSql( fnode, sqlquery, reltype );
    card := SQLUtil.SQLcard( ADR( sqlquery ) );
  ELSE (* Cannot avoid the temporary relation *)
    lrel := Eval( fnode );
    card:= CRDS.Card( lrel );
    CRDS.CloseRelation( lrel );
    ErrorHandler.Assert( CRDS.Done() );
  END;
  CPMSUtil.DropPredicate( fnode );
  RETURN card;
END Card;
```

DBXRefMy:

```
MODULE DBXRefMy;

...

TRANSACTION Integrity;
BEGIN
  IF ALL m IN DBXRef.ModuleRel
       (SOME i IN DBXRef.IdentRel
            (m.ModuleID = i.ID))  THEN
    Write("Module relation is consistent")
  ELSE
    Write("Module relation is inconsistent")
  END;
  Wait;
END Integrity;

TRANSACTION CreateInconsistency;
BEGIN
  INCL(DBXRef.ModuleRel,
       DBXRefType.Module{9999999, "Inconsistent", DBXRefType.defM});
```

```
END CreateInconsistency;

BEGIN
  INCL(ErrorHandler.debugSwitches, 11);
  Integrity;
  CreateInconsistency;
  Integrity;
END DBXRefMy.
```

Das `DATABASE` Modul für `DBXRef` ist:

```
DATABASE DEFINITION MODULE DBXRef;
IMPORT
  DBXRefType;

VAR                                  (*persistent database variables*)
  IdentRel        : DBXRefType.IdentRelT;
  ModuleRel       : DBXRefType.ModuleRelT;
  DependsOnRel    : DBXRefType.DependsOnRelT;
  DefModOccurRel,
  ImpModOccurRel  : DBXRefType.OccurRelT;
  IdentCt         : DBXRefType.ID;   (* highest ID key issued up to now. *)

  TRANSACTION InitDB;

END DBXRef.
```

Technisches
Die Demo sollte auf

```
/users/dbis1/DBPL/lib/sparc
```

gestartet werden, weil dann genau die Quelltexte des Laufzeitsystems verfügbar sind. Der Start der Demo selbst erfolgt mit:

```
make -f /users/dbis1/DBPL/test/dbplxref/Makefile demo
```

Ein fehlender Lockserver kann auf der dbis1 mit

```
make -f /users/dbis1/DBPL/test/dbplxref/Makefile server
```

gestartet werden.