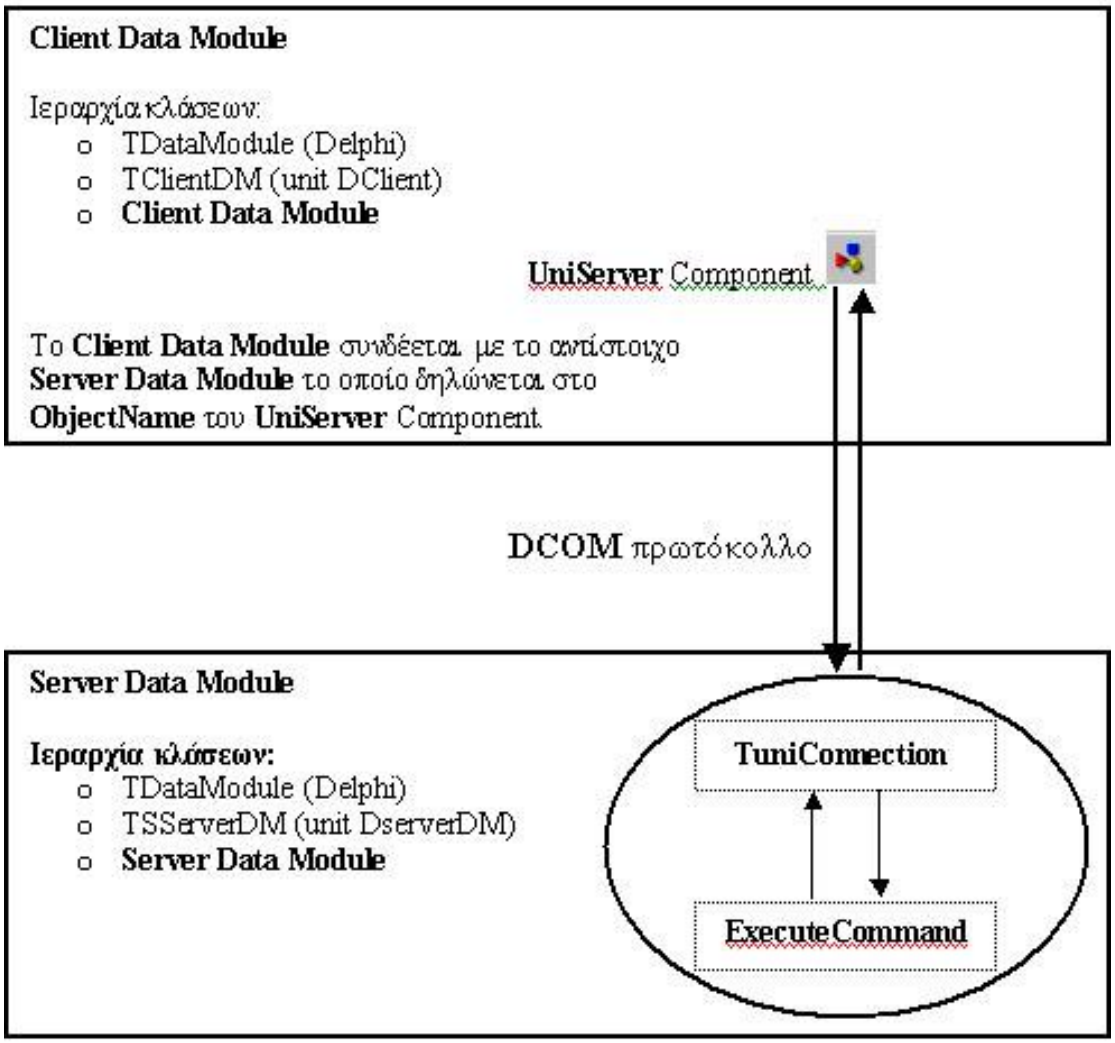
ΕΚΤΕΛΕΣΗ ΕΡΓΑΣΙΩΝ ΣΤΟΝ SERVER (EXECUTE COMMAND)

Γενικά

Για την ανάπτυξη εφαρμογών που απαιτούν την εκτέλεση εργασιών στον **Server** που ορίζονται δυναμικά από μια επικοινωνία Client-Server ακολουθείται μια προτεινόμενη διαδικασία υλοποίησης η οποία περιγράφεται παρακάτω. Η διαδικασία αυτή υλοποιείται με την επίτευξη μιας συγκεκριμένου τύπου σύνδεσης μεταξύ Client-Server που καθορίζεται από το class UniConnection, το οποίο ορίζεται αυτόματα κατά τη δημιουργία (creation) ενός Server Module. Η διασύνδεση μεταξύ του Client και του Server επιτυγχάνεται δια μέσου του component UniServer, το οποίο καθορίζει μια αμφίδρομη DCOM επικοινωνία μεταξύ ενός Client Module και ενός Server Module όπως περιγράφεται στο **σχήμα Ι**.



Σχήμα Ι: Επικοινωνία Client-Server, ExecuteCommand – TuniConnection

Για να επιτευχθεί μια επικοινωνία Client-Server χρησιμοποιώντας τα εργαλεία της Unisoft, δημιουργεί ο χρήστης δυο DataModule ένα για τον Client και ένα για το Server από το **Repository** του **Client** και το Repository του **Server** αντιστοίχως, ούτως ώστε να δημιουργηθεί αυτόματα όλη η ιεραρχία που περιγράφεται στο **σχήμα Ι** και να κληρονομηθεί και ο τύπος του TUniConnection. Εν συνεχεία αναγράφεται στο visual component **UniServer** του **Client DataModule** το όνομα του **Server DataModule**, στο πεδίο **ObjectName** του UniServer. Με αυτό τον τρόπο έχουν δοθεί επαρκείς οδηγίες για μια αμφίδρομη DCOM σύνδεση.

Έχοντας εξασφαλίσει μια αμφίδρομη DCOM σύνδεση μπορεί πλέον ο χρήστης να εκτελέσει συγκεκριμένες ρουτίνες και στον Client και στο Server να μεταφέρουμε δεδομένα με τρόπο που θα περιγραφεί παρακάτω. Αυτό που πρέπει να τονιστεί εδώ είναι ότι *όλη η διαδικασία της σύνδεσης παραμένει στο* ***background*** της εφαρμογής έτσι ο χρήστης δεν είναι υποχρεωμένος να γνωρίζει τη διαδικασία που εκτελείται στο παρασκήνιο, παρά να δηλώσει μόνο το όνομα της κλάσης του Server DataModule στο UniServer του Client.

Client (TExample – unit cExample)

```
Procedure TExample.DoUpdate;
begin
  ExecuteCommand(1,null);
end;
```

Server (TDExample – unit DExample)

```
function TDExample.ServerDMCommand(const CommandId: Integer; ParamValues: OleVariant): OleVariant;
begin
 inherited;
 Case CommandId of
   1: DoUpdateProcedure;
   2: DoOtherProcedure;
   ...
 end;
end;

procedure DoUpdateProcedure;
begin
  ...
end;
```

Παράδειγμα Ι: Η χρήση του ExecuteCommand και η σύνδεσή του με τον Server.

Η διασύνδεση του UniConnection με τη ρουτίνα την οποία θέλει ο χρήστης να εκτελεστεί στο Server γίνεται με το Event του Server DataModule **OnCommand**. Ενώ η εντολή κλήσης της ρουτίνας του Server από τον Client γίνεται απευθείας με την εντολή **ExecuteCom-mand**. Η ExecuteCommand καθώς και το event OnCommand του Server DataModule δέχονται δυο παραμέτρους, έναν ακέραιο (CommandId : Integer) και μια Olevariant μετα-βλητή (ParamValues : Olevariant). Στο **παράδειγμα Ι** υλοποιείται μια κλήση από τον Client μιας ρουτίνας που εκτελεί μια εργασία στον Server.

Αμφίδρομη επικοινωνία – Χρήση της ExecuteCommand σε διάφορες εργασίες

Η επικοινωνία μεταξύ Client και Server είναι αμφίδρομη. Αυτό σημαίνει ότι ο Server μπορεί πριν, μετά και κατά τη διάρκεια της εκτέλεσης κάποιας ρουτίνας που έχει κληθεί από τον Client να στέλνει διάφορα ερωτήματα ή να ζητάει περαιτέρω πληροφορίες για την έκβαση της εργασίας. Αυτό μπορεί να επιτευχθεί με τις τεχνικές που περιγράφονται στις δυο ενότητες που ακολουθούν (*Εκτέλεση batch εργασιών και Δήλωση πολλών διαφορετικών batch εργασιών στην ίδια ExecuteCommand*).

Το event **onCommand** επιστρέφει μια **OleVariant** αφού εκτελέσει την εντεταλμένη εργασία του. Η δυνατότητα αυτή επιτρέπει να χρησιμοποιήσει ο χρήστης την ιδιότητα **result** του συγκεκριμένου event (το οποίο δηλώνεται ως function) για να μεταφέρει πληροφορίες από τον **Server** στον **Client**. Το αντίστροφο (δηλαδή μεταφορά δεδομένων από τον **Client** στον **Server**) επιτυγχάνεται με τα argument που ορίζονται στο event onCommand. Η δήλωση του event έχει δυο παραμέτρους όπως έχει προαναφερθεί, τον ακέραιο **CommandId** και το OleVariant **ParamValues**. Τα δεδομένα που θέλουμε να μεταφερθούν περνούν δια μέσου της παραμέτρου **ParamValues**, η οποία μπορεί να δηλωθεί με τους τρόπους που προαναφέρθηκαν για τη δήλωση του result του event OnCommand.

Είναι σημαντικό να τονισθεί δε ότι η αλληλουχία μεταφοράς δεδομένων μεταξύ Client-Server έχει ως εξής: **μεταφέρεται αρχικά η πληροφορία από τον Client** και στη συνέχεια εκτελείται η ρουτίνα στο Server που καλείται από το event onCommand. **Αφού εκτελεστεί όλη η εργασία** τότε ο Server επιστρέφει τις πληροφορίες στον Client. Βλέπε **παράδειγμα ΙΙ**.

Client (TExample – unit cExample)

```
Procedure TExample.DoUpdate;
var Adata : OleVariant;
begin
 //Γέμισμα ενός πίνακα ClTable με τα αποτελέσματα ενός Query που τρέχει στον Server, με περιορισμό τα id που
 //φέρνει το Query να είναι μεταξύ 1411 και 1800.
 Adata := ExecuteCommand(1, VarArrayOf([1411, 1800, true]))
 if not VarIsNull(Adata)
  then ClientException('Δεν προέκυψαν στοιχεία')
 else ClTblData := Adata;
end;
```

Server (TDExample – unit DExample)

```
function TDExample.ServerDMCommand(const CommandId: Integer; ParamValues: OleVariant): OleVariant;
 var TempTbl : TUniTable;
begin
 inherited;
 TempTbl := TUniTable.Create(Self);
 TempTbl.Data :=  GetQueryData('Select ID, CODE, DESCR from Material where ID>= :1 and ID<= :2', 'ID;ID',
                                                    VarArrayOf([ParamValues[0], ParamValues[1]]));
 if ParamValue[2]
  then DoUpdateProcedure;
 Result := TempTblData;
end;

function DoUpdateProcedure;
begin
  ...
end;
```

Παράδειγμα ΙΙ: Η χρήση του ExecuteCommand και η σύνδεσή του Client με τον Server.

Εκτέλεση Batch Εργασιών

Στην προηγούμενη ενότητα περιγράφηκε μια επικοινωνία Client-Server όπου Ο Client μεταφέρει πληροφορίες στον Server και αφού εκτελέσει την εργασία επιστρέφει πληροφορίες στον Client. Η περίπτωση αυτή δεν καλύπτει περιπτώσεις που θέλει ο χρήστης να έχει αμφίδρομη επικοινωνία **κατά τη διάρκεια εκτέλεσης** της εργασίας ή στην περίπτωση που θέλουμε να **εκτελέσουμε την εργασία σε διακριτά στάδια**, π.χ. σαν πρώτο στάδιο την αρχικοποίηση, σαν δεύτερο στάδιο την κυρίως εργασία και σαν τελευταίο στάδιο την ολοκλήρωση.

Αυτές οι περιπτώσεις μπορούν να καλυφθούν με διαδοχικές χρήσεις της στοιχειώδους εργασίας που περιγράφηκε στην προηγούμενη ενότητα, όπου ως πληροφορίες θα περνιούνται και επιπλέον διαχειριστικές πληροφορίες.

Στο **παράδειγμα ΙΙ**, που περιγράφηκε προηγουμένως, μπορούμε να προσθέσουμε τις τρεις λειτουργίες που περιγράψαμε παραπάνω και επιπλέον να μας επιστρέφει μια πληροφορία για το σημείο στο οποίο βρίσκεται η εκτέλεση της εργασίας και να μπορούμε να αλλάξου-με την έκβαση της εργασίας δυναμικά. Στο **παράδειγμα ΙΙΙ**, περιγράφεται η υλοποίηση μιας τέτοιας διαδικασίας.

Το **παράδειγμα ΙΙΙ** περιγράφει τα εξής:

Στην αρχή αρχικοποιείται η διαδικασία τρέχοντας το Query με τις παραμέτρους που εισάγονται από τον Client και αφού τρέξει στον Server επιστρέφει το σύνολο των εγγραφών. Αν δεν υπάρχουν εγγραφές τότε ο Server επιστρέφει μια τιμή null, και αντιστοίχως ο Client τερματίζει τη διαδικασία, αλλιώς εμφανίζει μήνυμα που ζητάει την επιβεβαίωση συνέχισής της. Αν ο χρήστης επιβεβαιώσει, η διαδικασία συνεχίζεται, αλλιώς ολοκληρώνεται. Για να αναγνωρίζει ο Server το στάδιο της αρχικοποίησης, περνάμε μια παράμετρο που παίρνει τιμή 1 ή 0. Αν περάσει η τιμή 0 τότε τρέχει το Query και γίνεται η αρχικοποίηση, αλλιώς (αν έχει τιμή 1) διατρέχουμε τα δεδομένα που έχει φέρει το Query, και ανά κάθε γραμμή που διαβάζουμε στέλνουμε στον Client τα δεδομένα της γραμμής τα οποία τοποθετούνται σε ένα Client Table. Η διαδικασία αυτή συνεχίζεται μέχρις ότου τα δεδομένα να εξαντληθούν, αυτό το επισημαίνει ο Server στέλνοντας στον Client μια τιμή null και έτσι η διαδικασία ολοκληρώνεται.

Δήλωση Πολλών Διαφορετικών Batch Εργασιών στην ίδια ExecuteCommand (Συνθήκες)

Τέλος μπορεί να χρησιμοποιηθεί η ίδια ExecuteCommand για να εκτελεσθούν δυο ή και περισσότερες διαφορετικές λειτουργίες στον Server. Αυτή η ευκολία δίνεται με τη χρήση του CommandId. Μια περιγραφή του φαίνεται στο **παράδειγμα IV.**

Client (TExample – unit cExample)

```
Procedure TExample.DoUpdate;
var Adata : OleVariant;
begin
  Gauge.Progress := 0;
  //Γέμισμα πίνακα ClTable με τα αποτελέσματα ενός Query του Server με ταυτόχρονη ενημέρωση ενός progress
  //bar. Να μας ενημερώσει για το πλήθος εγγραφών, ζητώντας την επιβεβαίωσή μας για τη συνέχεια της εργασία.
  Adata := ExecuteCommand(0, VarArrayOf([1411, 1800, false, 0]));
  if not VarIsNull(Adata)
    then ClientException('Δεν προέκυψαν στοιχεία');
  else
    begin                                                          //ΑΡΧΙΚΟΠΟΙΗΣΗ
      if DlgAskYN(format('Πρόκειται να μεταφερθούν %d εγγραφές, επιβεβαίωση', [Adata])) = mrNo
        then Exit;
      Adata := 0;
      While not VarIsNull(Adata) do
        begin
          Adata := ExecuteCommand(1, VarArrayOf([1411, 1800, false, 1]));    //ΔΙΑΔΙΚΑΣΙΑ
          ClTbl.Insert(null, null, Adata[2], 1);
          Gauge.Progress := Adata[3];
          Gauge.Update;
        end;
    end;
    Adata := ExecuteCommand(1, VarArrayOf([null, null, true, 1]));     //ΤΕΛΟΣ
    Gauge.Progress := 100;
    Gauge.Update;
end;
```

Server (TDExample – unit DExample)

```
function TDExample.ServerDMCommand(const CommandId: Integer; ParamValues: OleVariant): OleVariant;
begin
  inherited;
  Result := null;
  Case CommandId of
    1: Result := DoUpdateProcedure(ParamValues);
    ...
  end;
end;

function DoUpdateProcedure(ParamValues: OleVariant) : OleVariant;
  var TempTbl : TUniTable;
begin
  Result := null;
  if ParamValues[3] = 0 then
    begin
      TempTbl := TUniTable.Create(Self);
      TempTbl.Data := GetQueryData('Select ID, CODE, DESCR from Material where ID>= :1 and ID<= :2', 'ID;ID',
                                                          VarArrayOf([ParamValues[0], ParamValues[1]]));
      Result := TempTbl.RecordCount;
    end
  else
    begin
      If not TempTbl.Eof
        then Result := VarArrayOf(TempTbl.FieldByName('ID').AsInteger,
                        TempTbl.FieldByName('CODE').AsInteger, TempTbl.FieldByName('DESCR').AsInteger, 100
                      * TempTbl.RecNo / TempTbl.RecordCount)
      else if ParamValue[2] then DoUpdateProcedure;
    end;
end;
```

Παράδειγμα ΙΙΙ: Η χρήση του ExecuteCommand και η σύνδεσή του με τον Server με ταυτόχρονη επικοινωνία του Client.

Client (TExample – unit cExample)

```
Procedure TExample.DoUpdate;
var Adata : OleVariant;
begin
  if CheckBox.IsChecked
    then CITbl.Data := ExecuteCommand(1, null)
  else CITbl.Data := ExecuteCommand(2, null);
end;
```

Server (TDExample – unit DExample)

```
function TDExample.ServerDMCommand(const CommandId: Integer; ParamValues: OleVariant): OleVariant;
 var TempTbl : TUniTable;

 procedure GetTempTableDataFromCustomer;
 begin
   TempTbl.Data :=  GetQueryData('Select ID, CODE, DESCR from Customer);
 end;

 procedure GetTempTableDataFromSupplier;
 begin
   TempTbl.Data :=  GetQueryData('Select ID, CODE, DESCR from Supplier);
 end;

begin
 inherited;
 TempTbl := TUniTable.Create(Self);
 Case CommandID of
 1 :  GetTempTableDataFromCustomer;
 2 :  GetTempTableDataFromSupplier;
 end;
 Result := TempTable;
end;
```

Παράδειγμα IV: Η χρήση του ExecuteCommand και η σύνδεσή του με τον Server με ταυτόχρονη επικοινωνία του Client για περίπτωση όπου εκτελούνται διαφορετικές λειτουργίες στο ίδιο ExecuteCommand.

Σύνοψη:

- Δημιουργείται μια DataModule στον Server.
- Δημιουργείται μια DataModule στον Client.
- Στην ιδιότητα ObjectName του component UniServer γράφεται το όνομα του DataModule του Server.
- Στο Event OnCommand γράφεται η ρουτίνα που επιθυμεί ο χρήστης να εκτελεστεί στον Server.
- Το Event OnCommand δέχεται δυο παραμέτρους, το CommandID και το ParamValues, όπου το CommandID αντιστοιχεί στη ζητούμενη εργασία του Server και το ParamValues είναι δεδομένα που περνιόνται από τον Client στον Server.
- Ο Server αφού εκτελέσει την εντεταλμένη του εργασία επιστρέφει μια OleVariant η οποία αποτελεί και την απάντηση του Server.

Για να δημιουργηθεί δυναμική επικοινωνία μεταξύ Client-Server φτιάχνουμε πολλές μικρές διαδοχικές "ερωταπαντήσεις" μεταξύ Client-Server.