# DTM System SCOP in a New Technological Generation

L. Molnar, J. Wintner, B. Wöhrer
Institute of Photogrammetry and Remote Sensing, The Vienna University of Technology, Vienna, Austria

ISPRS Commission IV, WG 4

**KEYWORDS:** DEM/DTM, Software Development, System Integration, GIS Integration

**ABSTRACT:**
According to the "Principles for a New Edition of the Digital Elevation Modeling System SCOP" [Mo92] presented at the Washington congress of ISPRS, a cross-platform, object oriented application frame has been created including GUI, graphics, and client-server components. Some major SCOP functionality has been transferred to this frame already, and new functionality has been added (in the first line Topographic Data Management [Ho96], and subdividing the model surface into regions). An introductory overview of the system is given, intended in the first line for qualified and experienced DTM/DEM specialists.

## 1. INTRODUCTION

At the Washington Congress of ISPRS, "Principles for a New Edition of the Digital Elevation Modeling System SCOP" have been presented [Mo92]. At the very least full ten man-years of R&D have been invested into realizing those principles. An application frame ('XX', see chapter 2) has been developed to dissolve "the contrast between merits and disadvantages of [university-based] research programs" as compared to commercial products - "concerning, in the first line, object oriented design, graphical user interfaces, interactive graphics, etc.". XX provides for client-server applications fully object oriented in their design; in practice, however, these applications realize an architecture that can be best classified as "componentWare" (see chapter 2).

Created under XX, the component "Topographic Data Selector and Editor" of SCOP.TDM (Topographic Data Management, the successor of TOPIAS) is now in beta-test [Ho96]. A pre-alpha version of SCOP.DTM integrating under XX some of the major components of the SCOP system, is demonstrated at the workshop "Advanced DTM Technology" at this congress.

The paper cited above [Mo92] is concentrating on R&D issues. This paper is one step closer to application: it is intended as a first introductory overview of SCOP under XX.

## 2. ARCHITECTURE

### 2.1 Versions

Beside the stand-alone versions, versions to be integrated into/with host environments are developed. Languages supported by the first edition(s) to become available are going to be English and German.

All versions support three modes of operation:
- GUI-mode (including also command-line entries),
- batch-mode to run unattended jobs, and
- slave-mode when controlled by host systems.

### 2.2 Object Orientation

A prediction that proved to be far too optimistic [Mo92]: "On a fine-grained level, with object oriented operating systems ... and with widespread processor support, distributed intelligence, and parallel processing - on this level, object oriented programming is not yet with us. For this, I think, even developers have to wait for maybe five years". - One cannot create any CAD class to inherit from AUTOCAD so to customize its functionality, and there is no ArcInfo consisting of distributed objects with virtual member functions - e.g. for digital terrain modeling that could be augmented by SCOP DTM class objects with members adapted correspondingly. Taligent Inc. was set up by IBM and Apple to create such a world - it seems to have failed, however, for technical and for commercial reasons [Co95]. "Object-oriented computing has failed. But component software ... is succeeding" - a headline of Byte Magazin (May 95). Parallel to this ISPRS congress, the first Component User's Conference is taking place in Munich, organized by the ComponentWare Consortium, supported by, and with speakers from IBM, Siemens, and other major companies; the issues are such as "Components: Another end to the software Crisis ?", "Evaluating Concurrency Models for CORBA Servers", "Survey of Data Management using CORBA", or "IBM OpenClass Component Framework".

Components in this sense are (large) modules with standard interfaces allowing to compose them into systems. Inside, components may be (and increasingly are) object oriented, but they may also be written in any other way, using, e.g., procedural languages and structured programming techniques - or they may be hybrid in their design. These components are in most cases controlled by means of "object broker techniques" - i.e. as entities showing object oriented features on their surface but being not necessarily object oriented in their internal architecture. - Beyond doubt, this is exactly what we termed "autonomous classes" [Mo92].

We suspect, however, that the situtation declared aggressively as "failure" of large-scale object orientation above is just due to impatience, to commercial and journalistic hectics. We tend to consider the "compnentWare contraRevolution" to be a transitory stage on the way toward consistent large-scale object orientation. But we are also in this transitory stage. The XX framework itself belongs to proper object orientation, it can address proper objects distributed over a network, but it is also suited to control independent server modules designed as ComponentWare.

## 2.3 Distributed Objects and Hybrid Components

XX consists of three application-independent components: a GUI-component (UX), a graphics-component (GX) and a communication- and system-controlling (supervisor) component (SX). All these are written in C++. Existing applications or modules (mainly in FORTRAN 77) act as "server components" running within separate processes.

The connection between these server components and UX/GX is established by so called *agents*. These are application specific components written in C++.

The general structure of an application built around XX is as follows:

```
                    I <--> UX,GX
Main <--> SX I
                    I <--> agents <-->  server components

<--------------- C++ -------------->        <---- hybrid ---->
```

A user action (e.g. a mouse click at some button "ISO") results in a message sent by UX to the ISO-specific *agent* which delegates the necessary computation task to the relevant server component(s). In a similar way, *agents* communicate among themselves by sending/receiving messages, thus composing the product (e.g. some view) as requested by the user and as corresponding to the current stage of processing.

Message-passing is controlled by the module SX, and it is performed by means provided by it. Message passing is asynchronous, so that intensive computation will not block the user-interface.

Message passing is built upon an Object Request Broker (ORB) which conforms to the CORBA 2.0 standard of the Object Management Group (OMG), an orginazation who's aims are the standardization and adoption of object technology (all leading companies have joined this group). Using an ORB allows for distribution of components on different machines in heterogeneous environments. The commercially available ORB libraries ORBIX and ORBELINE have been tested to build two versions of SX.

SX supervises the integrity of the running system and takes care of proper handling of any failures in it. One of its means to do so is to continuously supervise the proper closing of message loops.

## 2.4 Graphical User Interface

On the one hand, in the spirit of object orientation, users should communicate with the objects visible to them directly, and this communication should reflect the *state* of the objects. On the other hand, however, "development, and sometimes fashion, do not stand still, ..., further changes are following in rapid sequence, considering, e.g., the exploding importance of multimedia technology. It is, therefore, imperative to separate the GUI from the classes to be controlled by it. There is an interface necessary, somewhat similar to the resource file statements as originally formulated by IBM for the PM. Interpreting such statements, the [component UX] can compose the graphical user interface to appear on the screen in using the latest cross-platform development tools and libraries. This way, the user interface can be kept to correspond, though with severe compromise, to the fashion of the time" [Mo92]. The most recent event to proove this point is the appearence of WINDOWS 95 with a new GUI: such changes should not enforce re-writing too much code in large applications such as SCOP.

To create a solution for this dilemma took (and still takes), indeed, a major sacrifice of capacity in research and in programming by this institute. One is forced to be critical of the current state of software technology not providing proper solutions in this general-purpose realm. (A detailed analysis of this dilemma, and an elegant concept for a world-wide solution is given in [Co95]). Standards for GUI interfacing are plainly missing.

Here, a very short outlay of our solution follows, useful to any thoughtful user of SCOP (or of any other system) under XX:

- The *Main* module, when started by the user (or by some other application) starts the supervisor component (SX) of the XX frame, and transmits to it the description of the main GUI window of the application.
- SX starts the user interface component UX, and transfers to it the description of the main window.
- UX interprets (at run-time) the description of the main window, displays it on the screen, and waits for user action(s).
- According to the user actions, UX sends messages via SX to the agents concerned (known to it from the description it received from Main). If not yet running, SX starts the corresponding agent(s), and passes the message(s) to them.
- In many cases, agents are going to send to UX descriptions for subLevels of user communication, e.g. to collect parameter or other entries from the user; these are then interpreted and displayed by it. - In other cases, the agents will compose services of the hybrid server modules to products as requested by the user. For this, SX will start the corresponding server component(s), send it(them) some control information (e.g. directives to existing SCOP modules, performing the task in the background, running in batch mode).
- Graphics produced will be sent by the agent(s) to the GX component of XX (see below). Error messages, summaries etc. will be displayed by means provided by UX.

The rest can be guessed from the above, we hope.

Specifications for the user interface are written as uxDL ("user interface Definition Language) statements. uxDL is specified and created for UX. It has a syntax similar to that of C++. Semantically, it is kept as far as possible independent of any current GUI in describing the functionality requested in abstract terms such as 'Main', 'numVal', 'analogVal', 'label', 'stateSwitch', 'RGB_DIGITAL' etc. There is a series of operators provided for agents to add or to subtract an element to/from the GUI, to modify it, to make it accessible/nonaccessible, for inquiry, and so on. uxDL statements carry detailed specifications for relative and absolute object positioning, for events to cause messages to be sent, a string for short help and an entry in the help database, initial value(s) to be set, etc.

UX builds up a database of these specifications, and it keeps it up-to-date with regard to agent and user actions.

The part of UX to display the GUI objects on the screen is doing this via driver-objects dependent on some cross-platform GUI library - at this point from Zinc Software Inc. Displayed is a native GUI, i.e. one looking as OSF Motif, as WINDOWS 95, or as OS/2 etc., depending on the platform for the version of SCOP (or of some other application created using XX).

There are two more specifics we would like to mention: stateSwitches, and commandLine.

StateSwitches are GUI buttons specialized to switch object states. There goes the saying that *constructs not having states are no objects*. There is much truth to this: just functions do not have states. On the contrary, objects - i.e. data with corresponding methods (member functions) to manipulate them - can be, e.g., 'not yet read in', 'not yet displayed', 'displayed' etc. And the functionality inherent in objects can be configured - 'set up'. With OS/2 and with WINDOWS 95, GUIs appeared on the market using this type of user controls (there used to be others earlier, naturally); they set the standards how to do it, and users will get used to it: a right mouse click '*opens*' the object for to be set up (i.e. for parameter specification), and different other actions (carriage return on the active object, left double mouse click) will '*activate*' the object according to the current setup specifications. This second action can switch just two states, and only in one direction: off-to-on. This shortcomming is still rooting in procedural thinking: they say they are "running the object" - a contradiction in itself, not without humour. So we have been forced to create the 'stateSwitch', an element doing more than "running the object": it is capable to swith back-and-fort among object states via keyboard arrows, via pop-up menue, or via hitting carriage return versus blank "over" the *current* stateSwitch; and, naturally, stateSwitches are capable of opening the object on right mouse-click or as choice from the pop-up menue.

Shading understood as a large array of pixels (i.e. as data, as object) yields a good example for switching states. This complex object is controlled on the screen by a stateSwitch. A right mouse click at it will open a 'subLevel' (a window) with parameters for setting up the computation; in most cases, however, the default setup will suffice, and therefore, the opening (setup) action will not be needed. The states allowed for this switch are: OFF, DISPLAYED, FROZEN, EDIT ( OFF and DISPLAYED are self-explanatory; FROZEN means, that while editing the terrain data by interactive graphics, the shading should remain displayed but not re-computed on every change; EDIT means that pixels of the shading can be interactively identified and edited).

Concerning the commandLine: it yields the always-present option of command entries. These are composed of (shorthands) of the labels written onto/at the different objects on the screen. To take the above example for shading, the actions described there could be replaced by typing

```
shd, zenDist 30, dsp;//or the similar
```

This allows for fast and special ways for expert users, for batch capability, and for writing command procedures.

**2.5 Graphics**

The graphics component of XX: GX is realizing a subset of CAD funcionality, and at the same time a superset of it concerning the special needs of geocoded data. GX can overlap, edit, and in certain special respects also process both vector and raster (pixel) type data.

To facilitate fast identification of on-screen objects for interactive graphics, GX is building up a database of its own for the addresses and graphical characteristics of these objects. Otherwise, data are managed by *interface objects* - a solution coherent with strict distributed object technology (see chapter 3). Interface objects are derived from an abstract interface class specified by GX thus declaring (and in certain aspects also defining) functionality as needed by it - such as 'identify closest to (x,y)', 'data within area of interest', 'register/commit alteration by the user', etc. To perform such tasks, the interface object can access, e.g., some database via any network - or, at the other extreme of the scale, its own separate excerpt from it. The most primitive examples of such databases are intermediate ('ZWIFI') files; for them, as far as possible, the methods are written as declared in the abstract GX interface class. Naturally, based on ZWIFI files this solution is inefficient and not really capable of any active graphics functionality. A much more versatile case is some interface class based on TOPDB tables [Lo91]: here, manysided selection operations, and efficient identification are granted, thus allowing for editing the contents of tables without being forced to duplicate them in another form.

GX has the capability to refresh screen graphics locally, i.e. to re-draw only those screen regions effected by changes. This is very important for following in real time (or at least near-real time) any data editing actions by the user: re-calculating and updating the currently active screen contents (e.g. contour lines drawn over hill shading) should be done only within the area affected by the change(s). - To solve this, there is much to be done yet in the (hybrid) SCOP server components.

Beyond interactive graphics, GX is responsible for plotter and metafile related functionality.

## 3. INTEGRATION

The continuous struggle with (and often against) three or four related software systems so to integrate the best of their respective services into harmonous products is most typical of to-days production. Rather than offering some specialized service in perfect quality combined with flexible means of integration, practically all systems try to augment their central functionality by simple additions of their own - additions too simple to be sufficient for professional application.

To a considerable extent, this is also true of us. Realizing this, the structure of XX itself, and of applications created using it, are made such so to allow for manysided ways of integration - the scale reaching from file interfaces at its traditional extreme to interface objects at the other extreme - that of distributed object technology.

### 3.1 Interface Objects

One of the most versatile means for interfacing different components is via *interface objects*. This is true for components integrated under XX, and also for components of the software environment to be integrated into production systems.

Objects are data bundled with the methods operating on them. Interface objects are data to be exchanged bundled with the methods to perform import and export of them. In special cases further methods may be necessary such as answering inquiries, performing changes etc. Interface objects may be parts of a distributed object system, or they may be implemented as client/server components; in more traditional cases systems both on the import and on the export side will link (parts of) their methods and share or attach-and-release the corresponding database.

Methods on both the import and the export side will carry driver-type functions capable of communicating with the system on that side. These functions will be written using means provided by the system - e.g. library functions to access proprietary databases and/or to interpret internal data representation.

### 3.2 Replacing Components by Standard Systems

Although not without considerable effort, it should be realistic to re-write the driver-type functionalities in SX, UX and/or GX so to arrive at a really seamless integration with the most important standard systems (such as ArcInfo, Intergraph MicroStation etc. ). Re-writing the display-drivers of UX with the corresponding display library of another system will result in a user interface appearing to the user as part of that system. In case of GX, replacing the entire graphics functionality by that of another system could be reached this way. Certain changes to the XX-internal data communication and to its contents will also become necessary for this level of integration.

For this way of integration, most of the important software systems possess the necessary libraries to this date. These are in most cases traditional (procedural) libraries, sometimes with "callback functions". Integration will become much simpler with the standard systems and their libraries becoming object oriented.

Replacing the database of SCOP (TOPDB) by other DBMS systems is handled next (paragraph 3.3).

### 3.3 [TOP]SQL

Internally, this version of SCOP is storing data on tables managed by the topological and relational database management system TOPDB [Lo91]. TOPDB is addressed by statements formulated in TOPSQL - a subset of standard SQL with additional topologic/geometric datatypes and operators.

TOPSQL statements can be re-formulated in terms of standard SQL, yielding a slow but functioning solution for replacing TOPDB by standard RDBMSs.

Much simpler and much more efficient is this same way for those RDBMSs capable to handle geometry and topology in special ways. Recently, ORACLE corporation released a special version of its RDBMS with such extensions. ORACLE databases are widely used in our profession anyway; this extension will, as we believe, strengthen ORACLEs position. So it becomes specially simple for us to create version(s) of SCOP based directly on ORACLE databases.

### 3.4 Data Conversion

There remains to state that traditional data conversion will long retain its importance in bridging the gap between application systems - and therefore, we can and will not neglect creating such "bridges".

## 4. REGIONS

### 4.1 (Irregular) Algorithm Tiling

Starting with this version of SCOP, the terrain (or other) surface can be subdivided into *regions* (an expression suggested by Prof. Ackermann), or - as referred to in [Mo92] - into (irregular) *algorithm tiles*. There is more than one reason for this new organization of surface representation:

- different regions of the surface may be so heterogenous in their character that they need at the very least *different parameters* of representing them by the same algorithm (this also includes such parameters as grid-step or computing unit size); but more than this:
- regions with different character may need *different algorithms* better suited to their specifics; this can include special cases such as cement roads, areas to be interpolated as (horizontal) planes or as river bases, and so on;
- regions with different ways of *data acquisition* requiring special interpolation algorithms - e.g. as contour lines digitized, profiling or grid measurements, etc.;
- introducing *computationally intensive algorithms* such as the ones capable of true 3D is hardly feasible for large models; also, this is only necessary in specific areas. These can be declared then as regions;
- providing for a much more *flexible switching among different (competing) algorithms* than it would be possible for the entire model - e.g. for purposes of experimentation by the user ("what-if analysis").

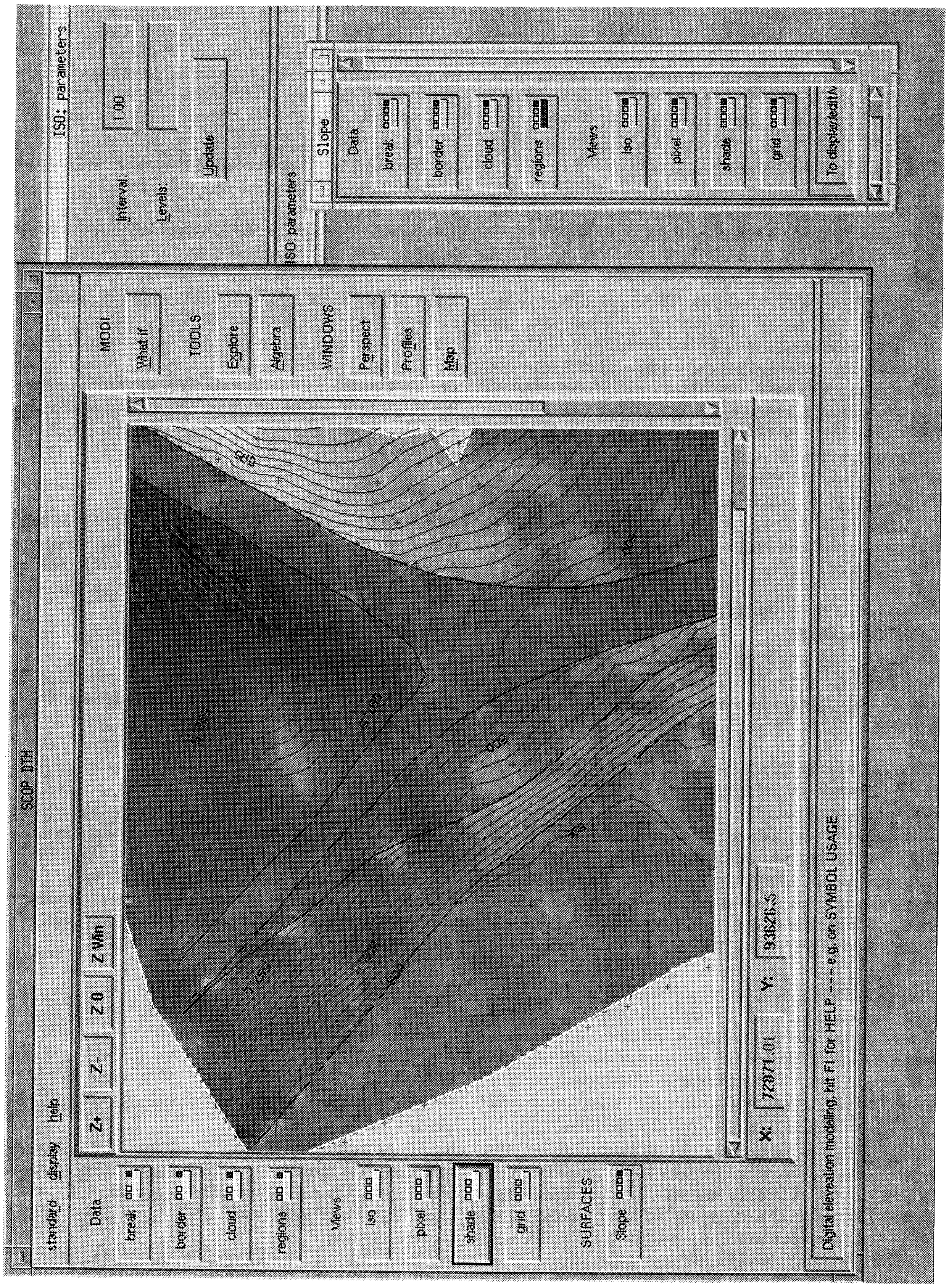

Fig 1. SCOP DTM under 'XX': main window, parameter window, window to control the function-model Slope

Regions are objects "logically encapsulating data ..., local co-ordinate systems, ... interpolation algorithms, private surface representation (e.g. hexagonal tiling, or analytical forms), functions to service inquiries concerning this surface (elevation, components of f' and f" at a given location, the isoline at a given elevation, ray tracing, etc.), and in some cases even hardware. They should employ parallelity in their co-operation ... [Mo92].

Realized is: the organization for regions, and creating separate SCOP DTM structures per region ("RDH").

### 4.2 (3D and Special) Enclaves

These are small regions with structures not necessarily belonging to the terrain surface as represented on topographic maps. Such structures are buildings, bridges, but details such as roads, sport fields etc. can also be classified as such. Small enclaves can just be neglected in certain cases (e.g. in creating maps in scales too small for representing them). If not small enough, the often just abstract topographic surface must also be specified for enclaves. Entire city models may be considered to be special and atypical enclaves.

There are no enclaves realized at this point.

## 5. PROCESSING

### 5.1 Interactive Mode

The aim of this description is to give the reader an idea of controlling SCOP under XX. This is a pre-alpha version, so any detail can change; the general idea remains proper, however.

*Fig. 1* shows a screen with the main window SCOP DTM, and two subWindows: ISO parameters, and Slope. On the left side of the main window, a series of *stateSwitches* is there to control *Data*, *Views*, and *SURFACES*. Clicking at the stateSwitch 'iso' with the right mouse button will open the subWindow for setting up parameters. There, interval and/or levels can be specified. A left click at the button 'Update' will result in taking over the values specified. It does not result in displaying isolines on the screen.

To display contourlines (isolines), the *state* of the switch has to be raised. This is done by left-clicking the stateSwitch, and hitting the right arrow or carriage return on the keyboard. The small bar on the stateSwitch will light up and blink in yellow - the blinking indicating the process running. The iso *agent* will be sent a message, it will request the contour lines at the agent for algorithm tiling. It then will initiate the interpolation for all regions within the area of interest (the area represented on the screen), then it will start SCOP.ISO to interpolate the contour lines. When all this is done, the iso agent will send the (address of the) interface object with the contours to GX. GX will display the contours. Finally, the iso agent will send a ready message to UX; the stateSwitch iso will sease blinking, the yellow light remains to indicate the state 'displayed' for isolines.

On the right side of the SCOP DTM window there are different buttons. 'What if' should serve to create one or more copies of the DTM so to experiment and to compare results. Explore, when 'opened', will present different means for "surface analysis and exploration", including, e.g., SCOP.SLOPE. Applying the latter will create a slope model; when created, it will appear on the left side under SURFACES (stateSwitch 'Slope'; when opened, the vertical window on the right side appears, showing for this model the same stateSwitches seen on the left side of the main window).

The button 'Map' under 'WINDOWS' will open a graphics window for editing output graphics.

Placing the mouse pointer over any symbol will display a short help string in the help bar along the lower edge of the corresponding window. F1 will yield detailed help.

For expert users, there is also a commandLine available. It controls all levels (main and subWindows) of processing. With commands as suited for it, command procedures can be written.

### 5.2 Compatibility

This coming version of SCOP integrates the modules of the most recent SCOP version as server components. They are capable of running in exactly the same "traditional" ways users of SCOP are accustomed to.

## 6. CONCLUSION

The main improvements in this presented version of SCOP are its openness toward future development and toward integration. Only in the second line would we mention the great improvement in processing control, and the introduction of regions.

At least half of the tasks formulated in [Mo92] remain open. There is much work underway. Closer to become applicable are:
- line networking (break lines, region limits)
- true 3D surface interpolation and representation
- intelligent overlapping/mixing of related views
- raster tools (a related reference: [Ri92])

## REFERENCES

[Co95] Cotter, S. Inside Taligent Technology. Addison-Wesley, 1995. ISBN 0-201-40970-4.
[Ho96] Hochstöger, F. Software for Managing Country-Wide Digital Elevation Data. International Archives of Photogrammetry and Remote Sensing, Commission IV, Vienna, 1996.
[Lo91] Loitsch, H., Molnar, L. A Relational Database Management System with Topological Elements and Topological Operators. Proceedings Spatial Data 2000, Dept. of Photogrammetry and Surveying, University College London.
[Mo92] Molnar, L. Principles For a New Edition of the Digital Elevation Modeling System SCOP. In: International Archives of Photogrammetry and Remote Sensing, Commission IV, Washington, D.C., 1992.
[Ri92] Rieger, W., Automated River Line and Catchment Area Extraction from DTM Data. Presented paper, ISPRS Congress, Washington, D.C., 1992.