SmartLink: An Agent for Supporting Dataflow Application Construction

Alexandru Telea, Jarke J. van Wijk

Eindhoven University of Technology,
Den Dolech 2, Eindhoven 5600 MB, The Netherlands,
alext@win.tue.nl, http://www.win.tue.nl/math/an/alext
vanwijk@win.tue.nl, http://www.win.tue.nl/cs/tt/vanwijk

Abstract. Visual programmable dataflow systems are an effective way to build a large class of visualization applications from existing software modules. However, the appeal of dataflow systems is often decreased as their users have to get familiar with libraries containing hundreds of different modules. Classical documentation systems such as hypertext or example suites are not always effective, as they lack the context of the user's questions and problems. We present a new visual dataflow programming assistant that is simple to use, offers context-sensitive help derived from the user's own behavior, and smoothly integrates in the effective point-and-click visual programming metaphor. We illustrate our approach with real-life usage examples.

1 Introduction

A recurring problem for scientific computing and visualization practitioners is the need to easily build new experimental applications for 3D data processing and visualization, imaging, or feature extraction. In many cases users need to build an entire suite of applications, e.g. when experimenting with different combinations of algorithms to explore a given dataset. Writing a new program for every trial application is prohibitively expensive in terms of time and convenience, especially for non experts in programming. Visual programming dataflow systems are a well known solution to this problem. Application building is done in such systems by picking visual representations of code components (or modules) in a module browser GUI and connecting them interactively to form a dataflow network. Users with little programming expertise can thus quickly create a suite of applications by reusing existing algorithms and data structures in a simple, intuitive way.

Although praised, visual dataflow programming systems are often less easy to use in practice than expected. Such systems usually come with large libraries offering hundreds of different modules spanning application areas as diverse as imaging, 3D rendering, CFD visualizations, charting, and volume visualization. The expected effectiveness of visual dataflow programming is noticeably diminished by the difficulty to find the 'right' modules to build the desired application among this large set of available choices. This difficulty is increased by the fact that the provided module set can freely grow with user programmed modules. Often users have had to rewrite existing modules simply since they were unable to locate them in the already existing large libraries. Schemes that address

this documentation problem such as online manuals and help agents have proven to be of a limited use in case of a large number of modules and users with limited expertise.

We have addressed this problem by devising the SMARTLINK help agent. SMARTLINK provides a new manner of assisting users visually in building dataflow applications by effectively exploiting their domain-specific knowledge and learning their preferences and interests. The concept presented here is a visual supplement to classical documentation browsers and can be used in a larger context than visual dataflow programming solely. We first discuss the limitations of the classical visual dataflow systems and of their help agents in Section 2. In section 3, we present the SMARTLINK concept. Section 4 extends the concept to object-oriented dataflow systems. Section 5 concludes the paper with a discussion of the method and ideas for future development.

2 Background

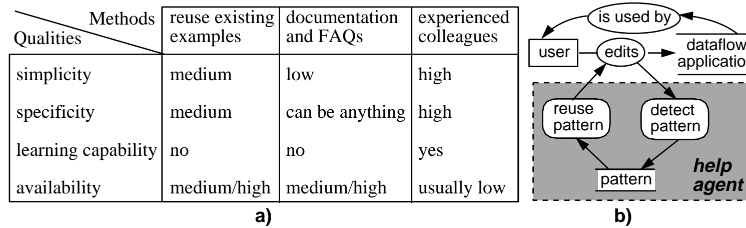
The dataflow programming paradigm is widely used by many systems, whether visual environments such as AVS or Express[4], Iris Explorer, or Khoros[5], or programming frameworks such as VTK[6], Open Inventor[2], or Java3D[3]. Application construction in such systems is an iterative task of finding the right modules and assembling them by connecting their inputs and outputs to create the desired dataflow network.

Whether visually programmable or not, such systems offer two main application building mechanisms. Firstly, a typing mechanism is provided for the modules' ports which enforces that only compatible ports can be connected. Object-oriented typed systems [6, 2, 8] take this a step further as OO typing automates some of the module input-to-output data conversions. End users are thus partially relieved of the burden of converting data explicitly by inserting conversion modules, which simplifies the network construction. Secondly, online help and documentation tools assist users in finding out information about a given module, such as the types and meaning of its ports, its operational semantics, an example of use, and hyperlinks to related modules.

Most of the improvement related to dataflow systems since their advent more than ten years ago has concentrated on areas such as external code integration and multi-language support [4, 8], object-oriented (OO) architectures [2, 6, 7], web-based integration and multiprocessing[4], and providing more modules and user interaction tools. However, the basic problem of assisting users in building dataflow applications has not received much attention. In practice, this task is mostly approached by a combination of the following three methods:

- modify an existing 'sample' dataflow network
- browse the online documentation to find out which module fits a given problem context
- ask human assistance (e.g. a more experienced colleague)

However effective, the above methods do not scale well in the context of a general-purpose dataflow system with hundreds of modules for various application domains, used by an unexperienced, possibly isolated user. Ideally, a visual dataflow programming help agent should posess several qualities, among which four are presented next (see also Fig. 1 a).



Qualities \ Methods	reuse existing examples	documentation and FAQs	experienced colleagues
simplicity	medium	low	high
specificity	medium	can be anything	high
learning capability	no	no	yes
availability	medium/high	medium/high	usually low

a)

Fig. 1. Visual programming help agents and qualities (a). The help agent in the network construction process (b)

Simplicity: Dataflow application construction should be simple to use and learn for novice users as well. Reusing an existing dataflow network by editing it is sometimes a viable solution. However, when no suitable example is available one is left with the tedious task of browsing huge amounts of module documentation. Using such documentation systems can be difficult, as these are usually built to reflect the module libraries' *structures*, and not their *use*. A known example for this are the OO component documentation browsers [3, 6]. To use such browsers, one must be familiar with OO notions such as inheritance and subtyping along which the documentation is structured. Understanding such documentation is often an all or none process, which is clearly undesirable for users that need only specific answers. Moreover, going back and forth between documentation browsing and network visual editing is clearly a disruptive process which one would like to avoid.

Specificity: The most ubiquitous questions of dataflow network building are not 'what does the module `ImageWarpFilter` do ?' but 'which module should I use to connect *this* data reader to *that* filter?' or 'what should I use to view *this* filter's data output?'. These questions have a clearly localized, context dependent nature. Consequently, generic documentation systems such as manual pages are not appropriate here, as these do not capture the question's context dependency. A documentation mechanism to capture more context is the frequently asked questions (FAQs) list. However, FAQs may also grow large, become less simple to use, and provide less specific answers. Generally speaking, the more specific and context-dependent the question is, the less probable it is to find the answer in general-purpose documentation.

Learning capabilities: As users build dataflow applications, they detect certain patterns such as the necessity or preference to use a certain filter in combination with some reader. Examples and documentation are however by definition static and thus can not reflect the user's accumulating knowledge unless they are rewritten as new information is learnt. Often users have no other alternative but manually create their own custom annotations and reuse them time and again.

Availability: In all the above, using the knowledge of another human is clearly the most effective way to learn to design new dataflow applications. However, experienced users from whom to learn are often not available. Availability is usually not a problem for automatic help agents such as documentation and examples.

Summarizing the above, we would like to have a simple to use, context-specific, intelligent automated agent to assist the construction of dataflow networks. Such an agent (the shaded area of the dataflow-like diagram in Fig. 1 b) should detect, capture, and exploit the user's behavioral patterns during network editing to assist the editing process. So far, these are performed explicitly by the human user during the learning process. Next, we present the construction and use of an automated network construction help agent.

3 The SMARTLINK Agent

The proposed solution is based on the observation that dataflow application building is mostly learnt by examples. Users find out that, for example, an `Actor`'s output should be connected to a `Viewer`'s input to view the actor, or sometimes to a `Writer` to write the actor to file. Next, when an `Actor` is present when building a new network, one infers a `Viewer` or a `Writer` might be needed, so a `Viewer` is instantiated and connected to the `Actor`.

We have exploited the above by constructing SMARTLINK, an agent that assists dataflow network building. Currently SMARTLINK is implemented atop of the VISSION general-purpose object-oriented visualization system [7]. However, the presented method can be easily added to any dataflow system. To describe the SMARTLINK agent, we first introduce a few terms. A *dataflow graph* consists of nodes, which represent module ports, and *links*, which represent data flows between these ports. A *path* is a sequence of links in the graph from one port to another. The *input* of a path is its first link's input port and its *output* is its last link's output port.

The SMARTLINK agent maintains a database able to store a set of links for all module ports in the dataflow system. These links represent the 'most used connections' into which that port is involved. For example, the 'output' port of an `Actor` might store two links, one to `Viewer`'s 'input' port, and one to `Writer`'s 'input' port. For every link an integer weight is stored that represents how many times that connection has been done. The database contains thus accumulated knowledge on system typical usage patterns. The database construction and the usage of the information it stores are described next.

3.1 Database Construction

The SMARTLINK database is initially empty. As the user employs the dataflow system to edit networks, SMARTLINK updates the database by looking at the port connect operations executed. When two ports p_1 and p_2 are connected, the two links $p_1 - p_2$ and $p_2 - p_1$ are looked up in the database. If found, the link's weight is incremented by 1, else a new link connecting the two ports and having weight 1 is inserted in the database. Figure 2 shows this for the example discussed above by depicting the database after an `Actor` was connected to a `Viewer` (a), then to a `Writer` (b), and finally again to a `Viewer` (c). Conceptually, these links form a separate *preferences graph* which indicates which module ports were connected throughout the system's utilization.

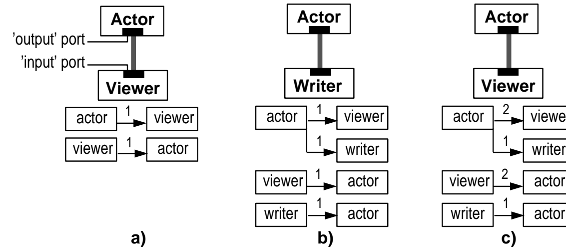


Fig. 2. SMARTLINK database for three network construction stages

3.2 Database Use

The information stored in the SMARTLINK database is used to assist network construction process by an interactive help agent. The purpose of the help agent is to answer the two types of questions discussed in Section 2, i.e. 'how can one connect two given modules?' and 'what can be connected to a given module?'. SMARTLINK addresses these two questions by its two operation modes called 'join' and 'cascading'. To illustrate these operation modes, we shall use a visualization example involving modules from the VTK library [6]. As described elsewhere[7], we have integrated VTK in VISSION such that its over 300 C++ components can be manipulated as visual icons in VISSION's network editor.

Join Mode: In this mode, SMARTLINK provides suggestions for the connection of two given module ports. We shall exemplify this by a VTK-based dataflow application for visualizing isosurfaces of a quadric function (Fig. 3). The pipeline consists of a scalar quadric function definition `VTKQuadric` sampled onto a regular grid by `VTKSampleFunction`. Isosurfaces are extracted from the sampled dataset by `VTK-ContourFilter`, mapped to geometric primitives by `VTKDataSetMapper`, then to drawable objects by `VTKActor`, and finally viewed in a `VTKViewer`. Additionaly, `VTKLookupTable`, `VTKProperty`, and `VTKDataSetInspector` control the colormap used, the actor's material properties, and the mapping of scalars to colors respectively.

Suppose now that all the user knows is that he wants to *visualize* the *isosurfaces* of a *quadric* function. He may start the network construction by creating a `VTKViewer`, a `VTKContourFilter`, and a `VTKQuadric` respectively, but he doesn't know how to continue. In a classic setup, he would have to browse the documentation or existing network examples to see what modules should be added to complete the network. In our case, however, all he needs to do is to connect visually `VTKQuadric`'s output with `VTKContourFilter`'s data input port. As the ports are not directly compatible, the system waits a short while (e.g. one second) to ensure that the connection attempt was not accidental, and then initiates a breadth-first search in the preferences graph from `VTKQuadric`'s output to `VTKContourFilter`'s data input port. The search is driven via a best-first heuristic by exploring links in decreasing weight order. Next, the system performs the same search in the opposite direction, i.e. from `VTKContourFilter`'s input to `VTKQuadric`'s output. More paths connecting the two ports may be found in this way, as the preferences graph is not symmetric with respect to the links' directions.

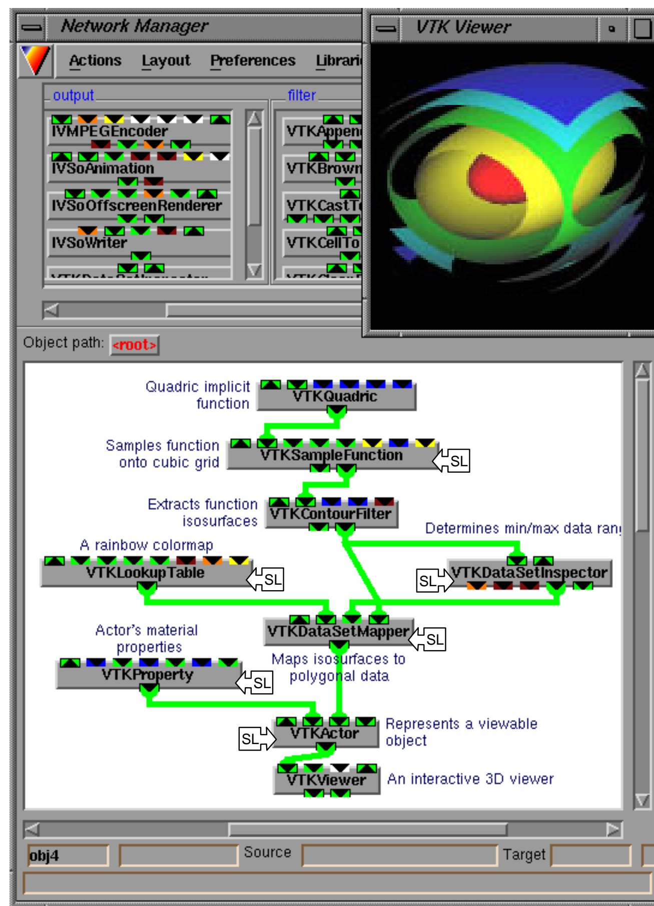


Fig. 3. VTK visualization example of quadric function isosurfaces

This is so since SMARTLINK may forget links from the preferences graph, as explained in Section 5. In our case, a path passing via `VTKSampleFunction` is found, as the system 'remembers' that this connection was done previously.

The path is displayed in a popup window (Fig. 4 a). If the user selects the path by clicking it, a `VTKSampleFunction` is created to connect `VTKQuadric` and `VTK-ContourFilter` (Fig. 4 b).

Next, we use the same procedure to connect `VTKContourFilter`'s output to `VTKViewer`'s input. Now three paths are found between the respective ports in the preferences graph. They are displayed in left-to-right decreasing order of their weights, computed as the sum of their respective links' weights divided by the square of the path length. The path weight computation favors often used (high weight sum) and short (low length squared) paths. Displaying the found paths in decreasing weight order ensures that the user sees the path he would probably best prefer first. In our context, the user will probably pick the second one. If this path is picked a few times, its links' weights will increase gradually until it will be found more important than the first one, when it will be

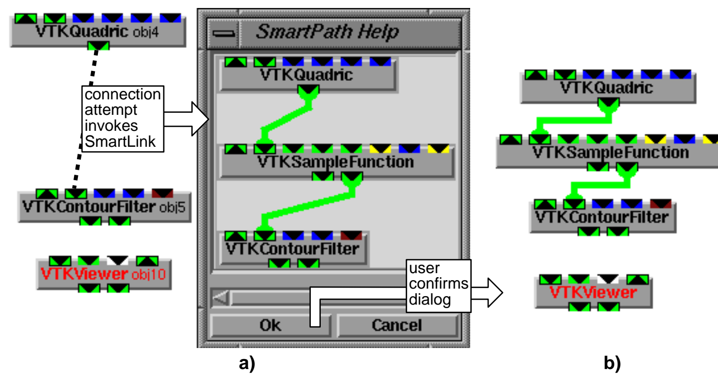


Fig. 4. Using SMARTLINK to connect a `VTKQuadric` to a `VTKContourFilter`

displayed first. The system has thus learnt from the user's behavior.

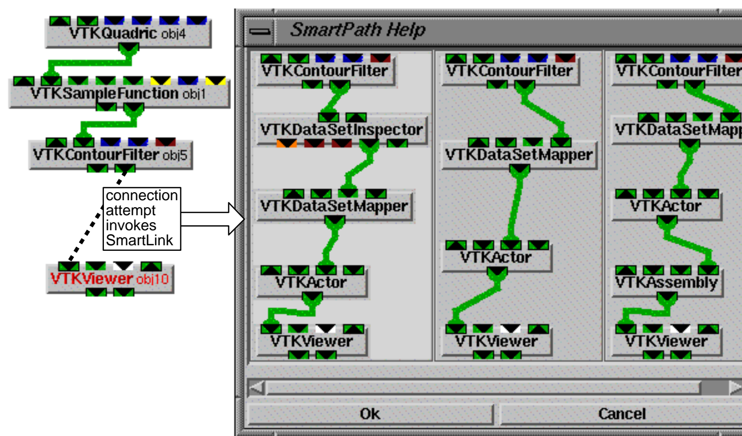


Fig. 5. SMARTLINK display of multiple paths

Cascading Mode: In this mode, SMARTLINK offers suggestions for modules that can follow from a given port. So far we have created the main part of the pipeline in Fig. 3 with only two mouse clicks. SMARTLINK offers also a second way to assist the network construction. In this so-called 'cascading mode', the user can simply click on any existing module port to find out which other modules can be connected there. For example, if the quadric visualization user didn't know how to view the isosurfaces output by `VTKContourFilter`, he could click on `VTKContourFilter`'s output to ask 'what could be connected here?'. A menu would pop up listing all the modules that `VTKContourFilter`'s output prefers to connect to, in decreasing order of the found links' weights. For example, to change the default colormap and material properties, two

clicks on VTKDataSetMapper's 'colormap' and VTKActor's 'properties' ports, respectively, are sufficient (Fig. 6 a-c).

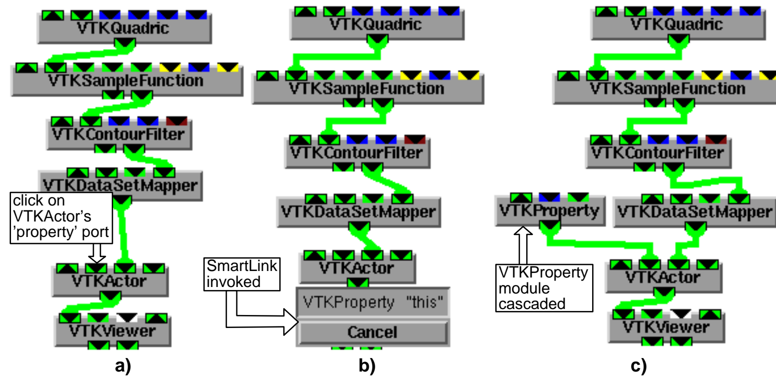


Fig. 6. SMARTLINK cascading operation. a) Port 'properties' of VTKActor selected. b) Popup menu with cascading option. c) Module VTKProperty cascaded

Overall, four mouse clicks are needed to construct the quadric visualization network from the three key initial modules. The six modules inserted automatically by SMARTLINK are shown marked with 'SL' in Fig. 3. Without SMARTLINK, this would have meant searching, instantiating, and connecting these modules in the right places. This would have taken longer even for an experienced user who knew exactly what and where to insert.

4 SMARTLINK and Object Orientation

VISSION is built on an object-oriented basis. VISSION modules are compiled C++ classes with inputs and outputs typed by user defined C++ types. Interpreted C++ is used for run-time scripting and type checking. Modules are organized as OO class libraries where derived classes inherit similar semantics and purpose, and specialize some feature [1]. For example, data reader modules inherit features such as input and output ports from a base class declaring the data reading interface and specialize the update operation. VISSION's OO foundations offer also an advantage regarding the SMARTLINK mechanism, as the next example shows.

VTK provides a class vtkImplicitFunction to represent all scalar functions $f(x, y, z) = 0$. This class declares an interface to evaluate the function, while subclasses such as vtkQuadric, vtkSphere, vtkPlane, and vtkImplicitBoolean define concrete quadric, spherical, planar, and boolean combinations of implicit functions respectively (Fig. 7). In dataflow terms, it is vtkImplicitFunction that declares the data output port via which vtkQuadric connected to the vtkSampleFunction in our visualization example. When this connection is made, SMARTLINK adds its information to vtkQuadric's output and also to the vtkImplicitFunction baseclass output (Fig. 7 b, steps 1-2). When SMARTLINK is subsequently used to assist connecting VTKSampleFunction's input, four possibilities are offered, namely vtkQuadric,

vtkSphere,vtkPlane, and vtkImplicitBoolean. Similarly, when the user requests help on connecting e.g. vtkPlane's output, SMARTLINK offers the VTKSampleFunction option, even though that module might have never used before (Fig. 7 b, step 3). The perceived effect is that the information learnt from the user propagates up in the class hierarchy and then down to the used modules' siblings. The SMARTLINK-OO combination is thus an effective, automatic way to answer questions such as 'I once used a VTKQuadric in some context - show me other modules I could use in the same context'.

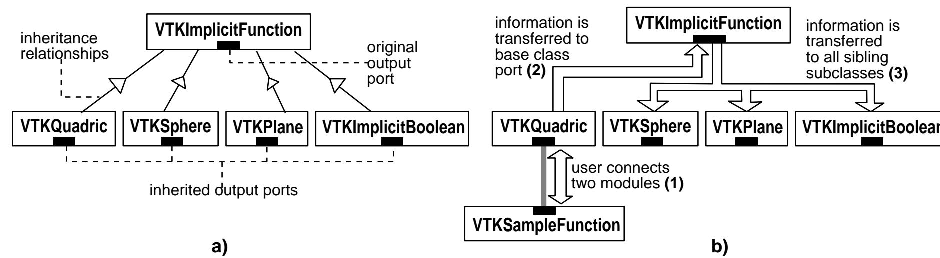


Fig. 7. Effect of object-orientation on SMARTLINK

5 Discussion and Future Work

We have presented SMARTLINK, an agent that makes visual dataflow programming easier and more attractive for non experienced users by exploiting the context information they inherently provide. The method does not replace, but augments, classical help media such as hypertext and examples, as follows. First, its use is simple (a single extra mouse click per usage) and does not disrupt the visual point-and-click network editing. Second, assistance is given automatically in a context specific manner, rather than in a generic, situation independent way. Third, the system learns from the user, so different users will be offered assistance that depends on their previous behavior. For example, an imaging researcher's database will contain information mainly on imaging modules, while that of a CFD engineer will contain mainly CFD-related links. To prevent database pollution with spurious information or its unbounded growth, we limit the number of links stored per module port to a small amount (e.g. 10) and manage them in a most recently used, highest weight first fashion. The system can thus 'forget' sparsely used links or links done by mistake. Finally, the method's combination with the object-oriented module libraries' structure is a simple but powerful way for generalizing from the learnt information. In contrast to most OO help systems, our method is simple to use as it hides the OO aspect completely from non OO-experienced end users.

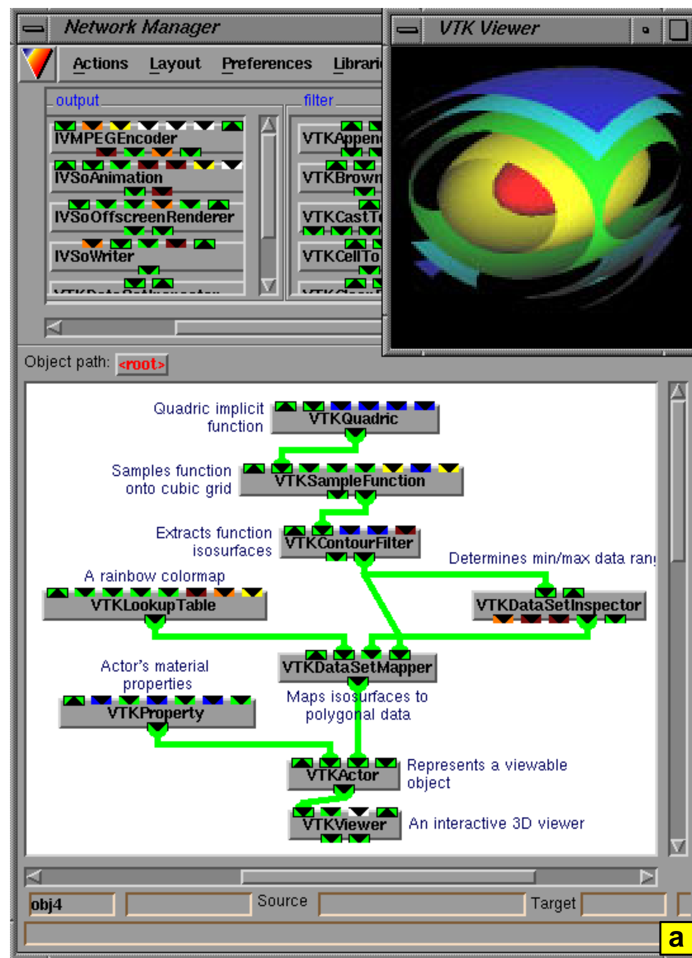
Users have found SMARTLINK very effective. Experienced users employed the cascading building method from a few 'key' modules to speed up network construction considerably. Novice users would copy the database of an expert in the field and get domain-specific online assistance for free, but block the agent's learning mode to prevent it learning from their mistakes. This emulates well the help of a specialized user,

as the database mirrors the human's system usage pattern. Database importing would interactively supplement documentation, tutorials, and examples.

SMARTLINK can be expanded in several ways. Different learning algorithms (i.e. weight increasing and path weight computation functions) may sample the user's behavior more effectively. A promising direction is to directly visualize the user behavior derived information. This would allow the identification of dataflow systems' usage patterns, and provide a compact, visual way to understand and work with large (dataflow) libraries. Finally, SMARTLINK could assist other programming tools that assemble typed components, such as C++ or Java visual compilers.

References

1. B. STROUSTRUP, *The C++ Programming Manual*, Addison-Wesley,1993.
2. J. WERNECKE, *The Inventor Mentor*, Addison-Wesley, 1993.
3. H. SOWIZRAL, K. RUSHFORTH, M. DEERING, *The Java3D API Specification*, Addison-Wesley, 1998.
4. C. UPSON, T. FAULHABER, D. KAMINS, D. LAIDLAW, D. SCHLEGEL, J. VROOM, R. GURWITZ, AND A. VAN DAM, *The Application Visualization System: A Computational Environment for Scientific Visualization.*, IEEE Computer Graphics and Applications, July 1989, 30–42. See also http://www.avs.com.
5. KUBICA, RASURE *The Khoros Application Development Environment*, Experimental Environments for Computer Vision and Image Processing, H.I Christensen and J.L Crowley eds, World Scientific, 1994.
6. W. SCHROEDER, K. MARTIN, B. LORENSEN, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Prentice Hall, 1995
7. A. C. TELEA, J. J. VAN WIJK, VISSION*: An Object Oriented Dataflow System for Simulation and Visualization*, Proceedings of the IEEE VisSym'99 Visualization Symposium, Springer, 1999
8. C. GUNN, A. ORTMANN, U. PINKALL, K. POLTHIER, U. SCHWARZ, *Oorange: A Virtual Laboratory for Experimental Mathematics*, Technical University Berlin. http://www-sfb288.math.tu-berlin.de/oorange/



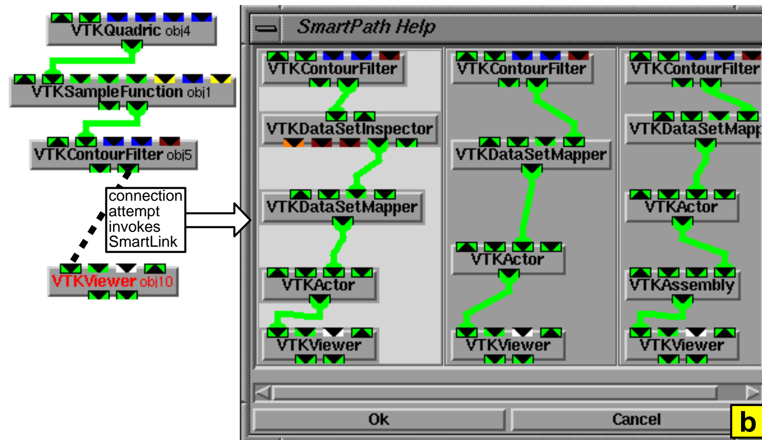


Fig. 8. VTK visualization example of quadric function isosurfaces (a) and SMARTLINK help agent invocation (b)