Visualizing Software for Telecommunication Services

Emden R. Gansner*
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932

John M. Mocenigo†
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932

Stephen C. North‡
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932

Abstract

An active research area in telecommunications concerns how to specify and control the addition of new services, such as call waiting or instant messaging, into existing software. One approach is to rely on a component-based architecture such as Distributed Feature Composition (DFC), by which a new service can be specified as a composition of primitive features over time. Formally, a communication episode is represented by a dynamic graph of software feature boxes, called a usage. This serves as the fundamental model for how services are invoked and how they interact with other services.

This paper, after providing some background on DFC, discusses a technique for visualizing the usages which arise through DFC specifications. With the visualization, users can monitor and validate service protocols and feature interactions in real time or through playback logs. The principal display component uses a novel variation of force-directed layouts for undirected graphs. The resulting graphical interface has become a principal tool for developers building services using DFC.

CR Categories: D.2.5 [Software Engineering]: Testing and Debugging—Debugging aids,Monitors,Tracing

Keywords: Algorithm animation, graph layout

1 Introduction

Traditionally, the introduction of telecommunications services, such as call waiting or multi-way calling, is plagued with problems. These arise due to the *ad hoc* nature in which services are designed, unexpected interaction with already existing features, and the intertwining of service and transmission features. All too often, when a new feature is implemented, it becomes apparent that its implementation has managed to interfere with or even break some older service. These difficulties only become worse when one considers services involving multiple media, networks and architectures.

Recently, the Distributed Feature Composition (DFC) architecture [Jackson and Zave 1998] has been developed to provide a platform for specifying the structured composition of modular features. DFC helps to solve many of the difficulties that are encountered in specifying telecommunication services. Of principal concern here, DFC models the invocation and interaction of communication services as an evolving graph of feature boxes.

The Building Box Communications project [Zave and et al. 2002] within AT&T Labs implements an IP-based service platform with DFC at its core. The project is exploring the value of DFC by building a set of advanced services, with the expectation of eventually allowing others to implement additional features and services. The project covers all aspects of telecommunication services, including "programming" of feature boxes, service provisioning, billing and system monitoring.

A principal component of the Building Box system is a user interface for viewing the evolution of a service in the context of network components and other services. Initially, the display aids the user during the process of specifying the service protocols and feature setups, Afterwards, having a visualization of the system assists in validating, and when necessary, debugging the specification. It also provides real-time monitoring of a running system.

The core feature of the interface is a novel display of the dynamic undirected graph representing the physical and logical features of the network. The vertices of the network naturally fall into two classes: external interface nodes and internal feature nodes. This suggests a "cloud and gateways" layout, with a boundary of interface nodes surrounding a cloud of feature nodes.

In the next section, we give an overview of DFC and its underlying graph model, along with a brief description of the Building Box project, and how it implements the DFC architecture. We then discuss the design of the interface component of Building Box, focusing on the desired aesthetic and interactive characteristics for aiding the user. This is followed by a description our implementation, a discussion of related work, and a section on possible extensions.

2 DFC and Building Box

The DFC architecture was developed to address the well-known problems of feature interaction in telecommunications. It provides a basis for the specification, verification, and implementation of telecommunication services aimed at avoiding these problems. In particular, the architecture was designed specifically for feature modularity, feature composition, and management of the complexity of feature interaction. It was also meant to aid the verification of various correctness properties, and to allow efficient implementation on a wide variety of physical networks.

DFC is an adaptation of the generic pipe-and-filter architecture to the telecommunication domain. It defines components for building features and services. Principal among these are *boxes*, which act as the primitive components of DFC. DFC boxes have well-defined interfaces and are granted structured access to operational data. A telecommunications service or feature is specified by the creation, modification and destruction of a graph of boxes in the context of a switch (or other computational node) containing other active features and services.

*e-mail:erg@research.att.com
†e-mail:john@research.att.com
‡e-mail:north@research.att.com

Figure 1 shows the model at a certain point in time of a simple DFC usage, with two telephones connected by a sequence of feature boxes. Line-interface boxes (LI) represent connections from the network to external components. This usage has a Call Forwarding/No Answer feature box which monitors signals from B and, after a given time, if no response is received, will tear down this call and construct a new call to a forwarding number.

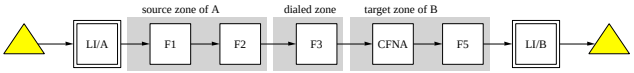


Figure 1: A typical usage sequence.

As depicted in the figure, there are two main box types. The first type are *interface boxes*, which demarcate the limit of the DFC logical network. They translate between DFC protocols and those of a particular hardware or software endpoint, such as a telephone, pager or other network connection. *Feature boxes* are the second type. They act as modular units for service construction, and are relatively transitory, existing only during the use of a feature. During the course of a communication episode, feature boxes are connected to form a graph joining a set of interface boxes. Signals are propagated through a chain of feature boxes, each of which responds based on the service it provides.

In the realistic situations DFC is meant to handle, a configuration often involves more than two interface boxes and multiple interacting feature sets. For example, Figure 2 shows a communication scenario involving four people and a variety of feature boxes including 800 number translation (800), a third-party call control feature (XC), call waiting (CW) and instant message alerting (IMA).

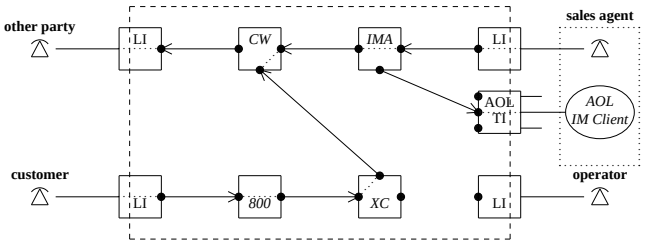


Figure 2: A DFC configuration.

Note that DFC only deals with the logic and protocols that establish services. A path in the graph from one interface node to another only indicates that the two nodes are involved in a communication. The actual communication may be handled by separate switching and transmission facilities.

Feature modularity is ensured by the rules governing feature boxes. As noted, the interfaces to boxes are well-defined and any two boxes are guaranteed to be composable. Boxes cannot base their behavior on assumptions about other boxes in current use. There are also strict rules governing the sharing of data between feature boxes.

The Building Box Communications project implements an IP-based service platform around DFC. The actual requirements of Building Box are defined independently of DFC, and focus on characteristics desirable for a service platform. These include such aspects as modular service creation, support for multimedia, separation of data and signaling paths, generality, and openness to third-party service creation. The working premise is that DFC supports most of these goals. The main challenge has been to map the logical, centralized architecture of DFC onto a physical, distributed system.

As Building Box is meant to be an operational system, it includes aspects and components that are not modeled by DFC, but rather support or augment the DFC architecture. Thus, the project addresses bandwidth and quality of service issues, as well as providing subsystems that handle billing, security, provisioning and monitoring. In this last category, the system provides a tool by which a user can view the DFC model as it executes. We will next describe the design and implementation of this tool.

3 Viewing Features

The Building Block Communications system provides a framework in which the designer of a feature can describe, via a specification language, how the feature is created and develops over time in terms of boxes. It soon becomes evident that to effectively understand how a feature operates, to debug a feature specification, or to monitor a running system, a user should be able to view the DFC model, and to see it dynamically as various features are invoked, interact with extant features, and then are revoked. Since the model is essentially an attributed graph, the natural visualization is a graph drawing, with the dynamic aspects of the model corresponding to the insertion and deletion of nodes and edges. Thus, a user should be able to view the system as an evolving graph. To preserve the user's "mental map" of the graph as it changes, consecutive drawings should be similar. To further preserve this context, the transition between graph states should also be shown by a smooth animation. And because the user needs to see the unfolding of events in response to real communication requests, the display should be able portray the system state in (near) real-time, at least for modest-sized graphs or subgraphs.

Once a graph has been chosen as the visual model, an important question is what type of drawing would be most effective. Feature graphs are usually sparse, with most edges part of long chains, but there appear to be no DFC-based or cognitive reasons for a non-traditional graph drawing of the boxes nor for a more constrained drawing such as a hierarchy [Sugiyama et al. 1981] or a circular drawing [Tollis and Xia 1995]. A standard "symmetric" drawing seems appropriate.[1]

However, in addition to feature boxes, the graph model has another distinct class of nodes, corresponding to interface boxes. The feature nodes are relatively transitory, existing only during while a feature is active during an episode. Interface nodes correspond to communications at the boundary of the system. These nodes are long-lived, modeling external hardware or software, and persist as features come and go. The main focus should be on the graph of feature boxes, but the interface boxes serve as a context for features, and are essentially separated from the subgraph of features.

Given these differences, we decided on a novel, specialized graph layout, with the interface nodes on a fixed periphery and the feature nodes appearing as a cloud in the interior. The feature nodes would be positioned using a general symmetric layout, which tends to distribute the nodes evenly, while attempting to keep edge lengths uniform and to display symmetries where they occur. This layout would be constrained to remain inside a boundary representing external connections. The interface nodes are constrained to the boundary, but are free to move on it based on their current connection to feature boxes.

The metaphor for this type of layout is that of a cloud and gateways. Figure 3 shows a typical layout of this type. The collection of feature nodes form an amorphous cloud, in our application representing a collection of features within the switch. All connections

[1] Though strictly speaking these layouts do not guarantee to display symmetries in a group-theoretic sense, the model treats edges as symmetric relationships and thus often reveals further symmetry. In any case, we follow the conventional name here.

from the nodes in the cloud to the outside world must pass through a gateway or interface node. Typically, there are no edges between interface nodes.

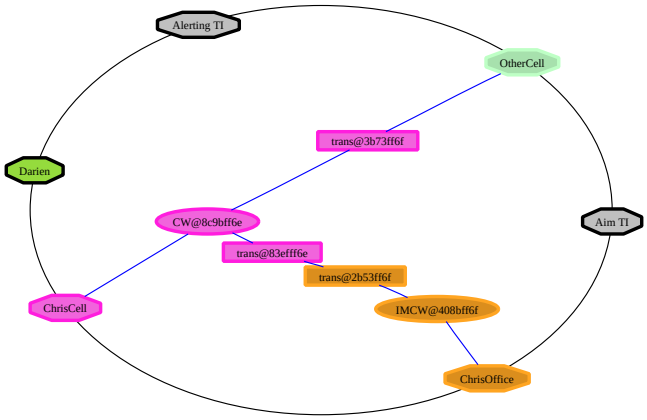


Figure 3: A typical cloud and gateway layout.

With the need for a basic symmetric layout within the cloud, the ability to translate incremental changes to the graph into incremental changes to the drawing, the flexibility to be modified to handle the periphery nodes, plus the performance constraints, we decided the layout could best be obtained by creating a variant of one of the family of virtual physical model algorithms [Fruchterman and Reingold 1991; Eades 1984; Kamada and Kawai 1989; Frick et al. 1995]. These represent a graph layout as a physical system of particles, with various forces[2] (spring, magnetic, gravitational, etc.) acting on the system. The nodes are then repositioned to minimize the cumulative forces or energy of the system. As an example, the graph shown in Figure 4 has the nodes in a random initial placement. Using a virtual physical model algorithm, the graph is rapidly untangled, yielding the layout shown in Figure 5.

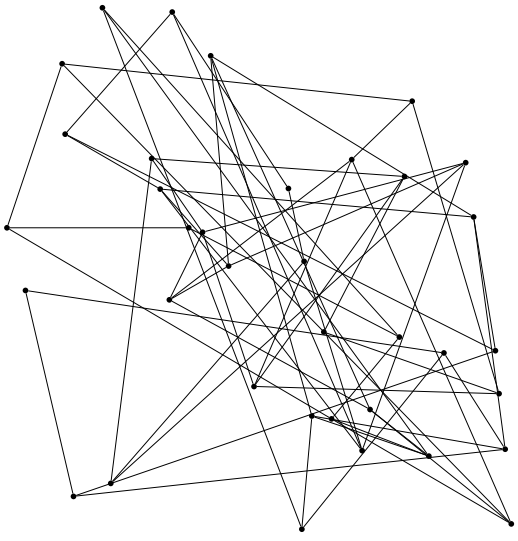


Figure 4: An initial graph layout.

[2]In practice, for reasons of both performance and aesthetics, the defined forces often deviate from physical reality.

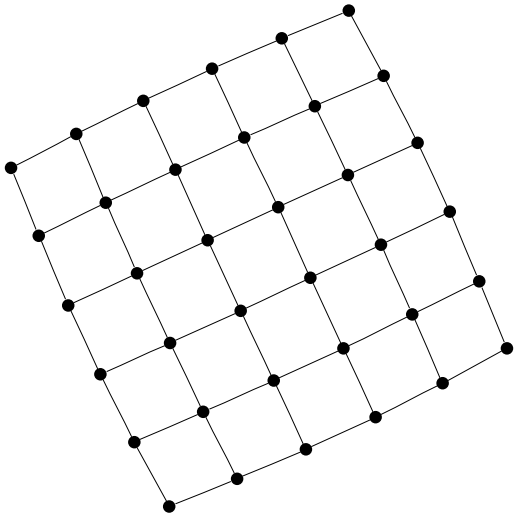


Figure 5: The final layout of the graph in Figure 4.

We started with a common variant on the Fruchterman-Reingold [1991] algorithm. In this technique, an inverse r^2 repulsive force is created between every pair of nodes, and spring forces model the edges. The nodes are placed in some initial position. The algorithm then iteratively calculates the aggregate force on each node and moves it accordingly. To aid convergence, the algorithm models a system "temperature," which restricts how far a node is moved at each step. Initially, the temperature is high enough to allow large motions by which the drawing can unfold to yield a reasonable global layout. Under some given cooling schedule, the temperature is gradually reduced, which progressively limits the movement of nodes while allowing fine-scale adjustments. The process is repeated until some termination criterion is satisfied, typically a maximum number of iterations or a small upper bound on the node displacements.

For simplicity, we use an ellipse for the gateway boundary to be an ellipse. Since we desire to maintain a fixed boundary, it has a fixed *width* and *height*. We set the initial temperature T_0 to $width/10$ and set the spring constant K to $(width \times height/|V|)^{\frac{1}{2}}$, where $|V|$ is the number of vertices in the vertex set V. If d is the distance between two nodes, we set the repulsive force between them to

$$F_r = K^2/d$$

and they have an attractive force of

$$F_a = d^2/k$$

if the nodes are connected by an edge. A simple linear cooling schedule is adequate.

Rather than have interior nodes stopped by contact with the boundary, we chose to effect repulsion. The algorithm was altered so that during the update phase, when the nodes are repositioned, the new constraints are enforced. Basically, we limit node movement by temperature [Fruchterman and Reingold 1991]. We then compute the elliptical radius, i.e., if a node is at (x,y), we calculate

$$r = (x^2/a^2 + y^2/b^2)^{\frac{1}{2}}$$

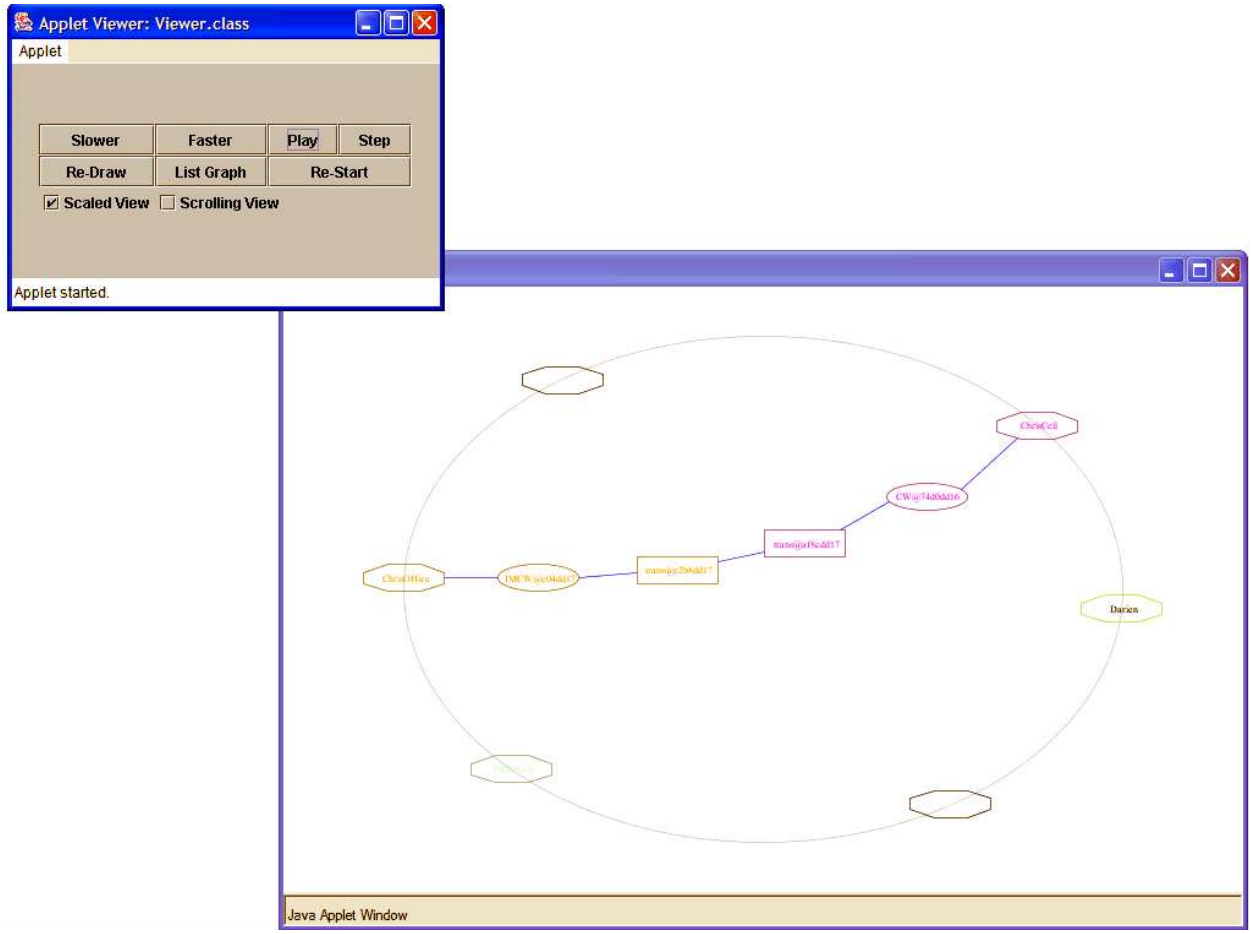


Figure 9: User interface controls

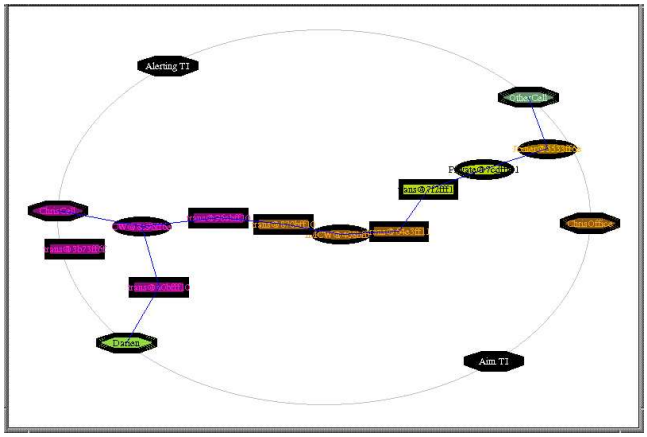
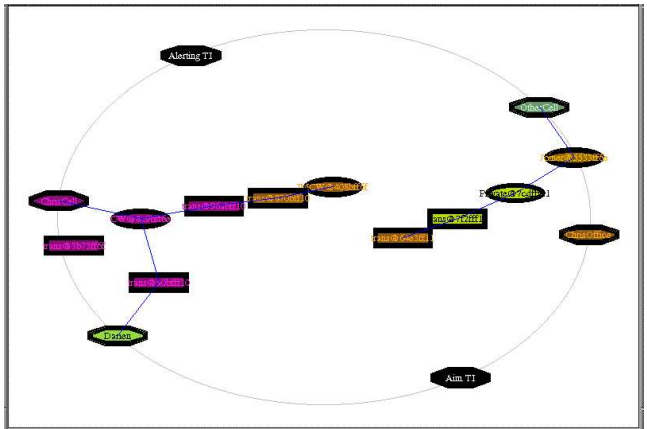


Figure 7: After removal of an edge



Unfortunately, this results in large jumps during the early steps. As interesting as this might be to implementors, it is irrelevant and distracting to users. Instead, we employ a simple, linear interpolation between consecutive layouts, using a fixed number of frames. (The number of frames can be set as a parameter.) After the graph has been laid out and each node assigned a new position, the displacement vector from the old to the new position is divided by the number of frames. For each frame, the node is moved by this increment, and the graph is redrawn, with the edges determined by their endpoints. Double buffering eliminates any flashing and, with a reasonable number of frames, provides smooth animation.

The prototype required about 1500 lines of Java, distributed over 32 classes. Much of the code is only involved with input, event parsing and output.

5 Related Work

There has been extensive work on using numerical solutions of a physical model to position the nodes of a graph. Some of this work is implicit in the techniques developed for various scientific and engineering disciplines. In some cases [Kruskal and Seery 1980] the use is explicit. Eades [1984], Kamada and Kawai [1989], Davidson and Harel [1996] and Fruchterman and Reingold [1991] provide an explicit starting point for these techniques in the graph drawing community.

These techniques are quite flexible and have been extended to handle various constraints. He and Marriott [1997] consider simple

linear constraints, while Kamps et al. [1996] incorporate positional constraints, such as placing a particular node above and to the right of another. Sugiyama and Misue [1995] use magnetic forces to achieve directionality. Dengler et al. [1993] propose a general constraint model, in which a linear combination of various constraint functions is minimized. In theory, this approach could be used to generate a cloud and gateways layout, but it is unclear how this would be done in practice, especially in a simple and computationally inexpensive manner. Similarly, simulated annealing [Davidson and Harel 1996] and genetic algorithms [Frick et al. 1996] are sufficiently general-purpose to be used, but their performance is unlikely to be acceptable.

Concerning the specific geometric constraint of laying out nodes on a smooth curve or surface, there has been theoretical work on the possibility and complexity of embedding graphs onto certain curves and surfaces [Tamassia and Tollis 1991; Kisielewicz and Rival 1995; Kratochvil and Przytycka 1995]. Dostry [1996] shows that many constraints, especially embedding nodes on a smooth surface, can be formulated using the same model as the unconstrained case. He also describes classical numerical analysis techniques which can be used to speed up the solutions. Even here, the complexity is $O(n^2)$ or $O(n^3)$, and the machinery is phenomenal.

There are few published references to applications using graph drawing with nodes constrained to lie on curves. The work of Brandes et al. [1999] provides one use in the social sciences. In this case, nodes are deterministically assigned to one of a set of concentric circles. Additional adjustments in position are computed using the algorithm of Kamada-Kawai [1989] with a node constrained to its circle.

The problem of constraining nodes to lie inside a region was considered early on. The technique of modeling the boundary as a potential energy barrier was considered by both Fruchterman and Reingold [1991] and Davidson and Harel [1996], and has since been applied by others [Dodson 1995]. This can be useful, but there is still the possibility that a particularly strong impulse could push a node outside the region. As in our work, these other approaches usually fall back on tracing the path of a node and preventing its escape.

Moreover, online graph drawing and other dynamic layout problems have been studied extensively. Misue et al. [1995] identified the central issue of preserving a user's *mental map*, and proposed maintaining the horizontal and vertical order of layout objects. Recent work explores the problem of smoothly animating transitions between discrete layout states that are computed by some external process, including situations where the successive states differ greatly [Friedrich and Houle 2002; Diehl and Görg 2002]. Progress has also been reported in dynamic algorithms for the main graph layout families. Physical models incorporating time lend themselves naturally to a dynamic interpretation [Huang et al. 1998]; in contrast, new techniques are introduced for dynamic *k*-layered and orthogonal drawing [Biedl and Kaufmann 1997; Brandes and Wagner 1998; Shieh and McCreary 2000; North and Woodhull 2002]. Further experimental evaluation and comparison of these algorithmically specialized techniques would be illuminating. Nevertheless, our viewer hews close to a basic incremental spring embedder model.

6 Conclusions and Future Work

We have described a visualization technique appropriate for system software that models telecommunications features and their behaviors as an attributed graph. The visualization allows the user to monitor how features are invoked and interact with each other in real time, and preserves the user's temporal context through incremental layout and animation. The drawing reflects the underlying model's division of nodes into the two categories of exterior interface and interior feature nodes, with the features constituting an unstructured core subgraph.

The system is now used daily by developers of the AT&T *VPLUS* project, a voice-over-IP system built on top of Building Box. Developer comments indicate that the visualization is absolutely fundamental in the development, debugging, administration and operation of the system. When questions arise about what is happening within the system, the tool of choice is to monitor the system visually using the graph representation.

Several enhancements and variations suggest themselves. One problem is that graph layouts made by the technique described may have unnecessary edge crossings. Crossings are especially distracting when a simple transposition of two adjacent interface nodes would remove them. Efficient crossing reduction heuristics usually rely on some additional graph structure, such as a layering of nodes. One can perform crossing reduction on general graphs by approaches like simulated annealing [Davidson and Harel 1996], but this is too slow for animation desired in our application. Local 2-swaps are another possible approach.

Also, note that the interface nodes are repositioned on the periphery at each step in the iteration. One could allow the layout to develop unconstrained through all the iterations, projecting back into the ellipse only at the final step. This might reduce the number of edge crossings as a product of the force-directed placement, which in its basic, unconstrained form, typically does a reasonable job of avoiding crossings in sparse graphs.

Another potential improvement is to introduce hysteresis into the node placement, so that if a node's new position is close to its old position, it is left unchanged.

Although the underlying changes to the graph occur by discrete events, adding and deleting edges and nodes, an entire communication episode might better be viewed as a continuous and continuing process. This suggests that, instead of showing a sequence of distinct graph drawings, one morphing into the next, we might interleave the graph update, layout and animation processes. Thus, the animation would be continuously changing to reflect continuously changing layouts. This approach has the additional benefit of telescoping the underlying sequence of events, helping the display keep pace with changes to the graph. This is of increasing importance as the size of the graph increases.

Finally, the cloud and gateways model should be extended to multiple clouds and multiple levels of nesting.

Acknowledgments

We thank our colleagues, Christopher Ramming and Gregory Bond, whose work on Building Box Communications motivated our work on this layout, and whose comments and suggestions helped us refine the implementation. We also thank Ramming for the use of Figure 2.

References

BIEDL, T. C., AND KAUFMANN, M. 1997. Area-efficient static and incremental graph drawings. In *European Symposium on Algorithms*, 37–52.

BRANDES, U., AND WAGNER, D. 1998. Dynamic grid embedding with few bends and changes. In *ISAAC: 9th International Symposium on Algorithms and Computation*.

BRANDES, U., KENIS, P., AND WAGNER, D. 1999. Centrality in policy network drawings. In *Proc. Symp. Graph Drawing GD'99*, Springer-Verlag, Berlin, J. Kratochvíl, Ed., vol. 1731 of *Lecture Notes in Computer Science*, 250–258.

DAVIDSON, R., AND HAREL, D. 1996. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics 15*, 4, 301–331.

DENGLER, E., FRIEDELL, M., AND MARKS, J. 1993. Constraint-driven diagram layout. In *Proc. IEEE Symp. Visual Languages*, 330–335.

DIEHL, S., AND GÖRG, C. 2002. Graphs, they are changing – dynamic graph drawing for a sequence of graphs. In *Proc. Symp. Graph Drawing GD'02*, S. G. Kobourov and M. T. Goodrich, Eds., vol. 2528 of *Lecture Notes in Computer Science*, 23–30.

DODSON, D. 1995. Comaide: Information visualization using cooperative 3d diagram layout. In *Proc. Symp. Graph Drawing GD'95*, Springer-Verlag, Berlin, F. Brandenburg, Ed., vol. 1027 of *Lecture Notes in Computer Science*, 190–201.

DOSTRY, D. I. 1996. *Some three-dimensional graph drawing algorithms*. Master's thesis, U. Newcastle.

EADES, P. 1984. A heuristic for graph drawing. *Congressus Numerantium 42*, 149–160.

FRICK, A., LUDWIG, A., AND MEHLDAU, H. 1995. A fast adaptive layout algorithm for undirected graphs. In *Proc. Symp. Graph Drawing GD'94*, R. Tamassia and I. Tollis, Eds., vol. 894 of *Lecture Notes in Computer Science*, 388–403.

FRICK, A., KESKIN, C., AND VOGELMANN, V. 1996. Integration of declarative approaches. In *Proc. Symp. Graph Drawing GD'96*, S. North, Ed., vol. 1190 of *Lecture Notes in Computer Science*, 184–192.

FRIEDRICH, C., AND HOULE, M. E. 2002. Graph drawing in motion II. In *Proc. Symp. Graph Drawing GD'01*, P. M. et al., Ed., vol. 2265 of *Lecture Notes in Computer Science*, 220–231.

FRUCHTERMAN, T., AND REINGOLD, E. 1991. Graph drawing by force-directed placement. *Software – Practice and Experience 21*, 11, 1129–1164. also as Technical Report UIUCDCS-R-90-1609, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1990.

GANSNER, E., AND NORTH, S. 1999. Improved force-directed layouts. In *Proc. Symp. Graph Drawing GD'98*, Springer-Verlag, Montreal, Canada, S. Whitesides, Ed., vol. 1547 of *Lecture Notes in Computer Science*, 364–373.

GANSNER, E. R., KOUTSOFIOS, E., NORTH, S. C., AND VO, K.-P. 1993. A technique for drawing directed graphs. *IEEE Trans. Software Engineering 19*, 3, 214–230.

HAREL, D., AND KOREN, Y. 2002. Drawing graphs with non-uniform vertices. In *Proc. Working Conf. Advanced Visual Interfaces AVI'02*, 157–166.

HE, W., AND MARRIOTT, K. 1997. Constrained graph layout. In *Proc. Symp. Graph Drawing GD'96*, S. North, Ed., vol. 1190 of *Lecture Notes in Computer Science*, 217–232.

HUANG, M. L., EADES, P., AND WANG, J. 1998. On-line animated visualization of huge graphs using a modified spring algorithm. *Journal of Visual Languages and Computing 9*, 6 (Dec.), 623–645.

JACKSON, M., AND ZAVE, P. 1998. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Trans. Software Engineering 24*, 10 (Oct.), 831–847.

KAMADA, T., AND KAWAI, S. 1989. An algorithm for drawing general undirected graphs. *Information Processing Letters 31*, 1 (Apr.), 7–15.

KAMPS, T., KLEINZ, J., AND READ, J. 1996. Constraint-based spring-model algorithm for graph layout. In *Proc. Symp. Graph Drawing GD'95*, F. Brandenburg, Ed., vol. 1027 of *Lecture Notes in Computer Science*, 349–360.

KISIELEWICZ, S. M. H. A., AND RIVAL, I. 1995. Upward drawings on planes and spheres. In *Proc. Symp. Graph Drawing GD'95*, Springer-Verlag, Berlin, F. Brandenburg, Ed., vol. 1027 of *Lecture Notes in Computer Science*, 277–286.

KRATOCHVIL, J., AND PRZYTYCKA, T. 1995. Grid intersection and box intersection graphs on surfaces. In *Proc. Symp. Graph Drawing GD'95*, Springer-Verlag, Berlin, F. Brandenburg, Ed., vol. 1027 of *Lecture Notes in Computer Science*, 365–372.

KRUSKAL, J., AND SEERY, J. 1980. Designing network diagrams. In *Proc. First General Conf. on Social Graphics*, 22–50.

LEE, W., BARGHOUTI, N., AND MOCENIGO, J. 1997. Grappa: A graph package in Java. In *Proc. Symp. Graph Drawing GD'97*.

MISUE, K., EADES, P., LAI, W., AND SUGIYAMA, K. 1995. Layout adjustment and the mental map. *J. Visual Languages and Computing 6*, 2, 183–210.

NORTH, S. C., AND WOODHULL, G. C. M. 2002. Online hierarchical graph drawing. In *Proc. Symp. Graph Drawing GD'01*, P. M. et al., Ed., vol. 2265 of *Lecture Notes in Computer Science*, 232–246.

SHIEH, F.-S., AND MCCREARY, C. L. 2000. Clan-based incremental drawing. In *Proc. Symp. Graph Drawing GD'00*, J. Marks, Ed., vol. 1984 of *Lecture Notes in Computer Science*, 384–395.

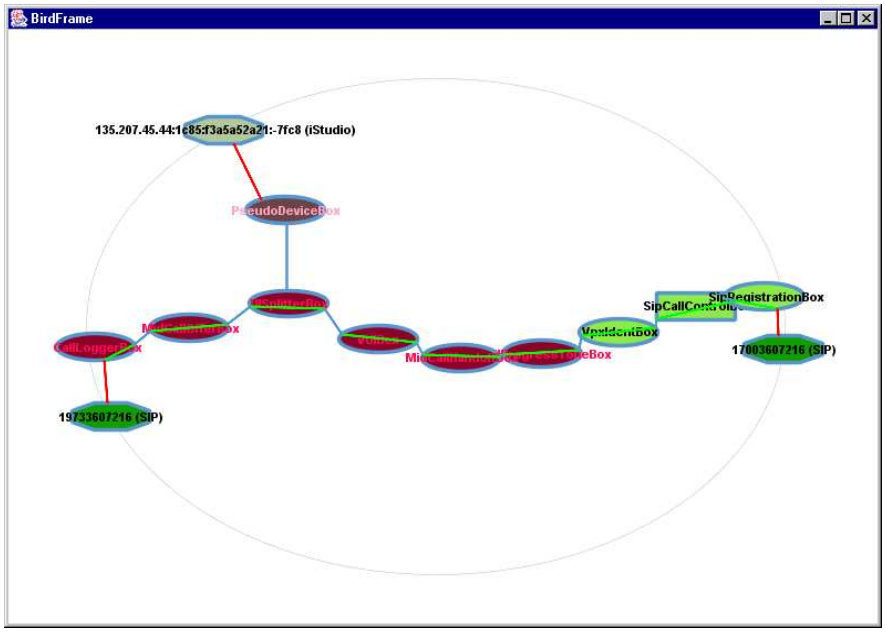
SUGIYAMA, K., AND MISUE, K. 1995. A simple and unified method for drawing graphs: Magnetic-spring algorithm. In *Proc. Symp. Graph Drawing GD'94*, R. Tamassia and I. Tollis, Eds., vol. 894 of *Lecture Notes in Computer Science*, 364–375.

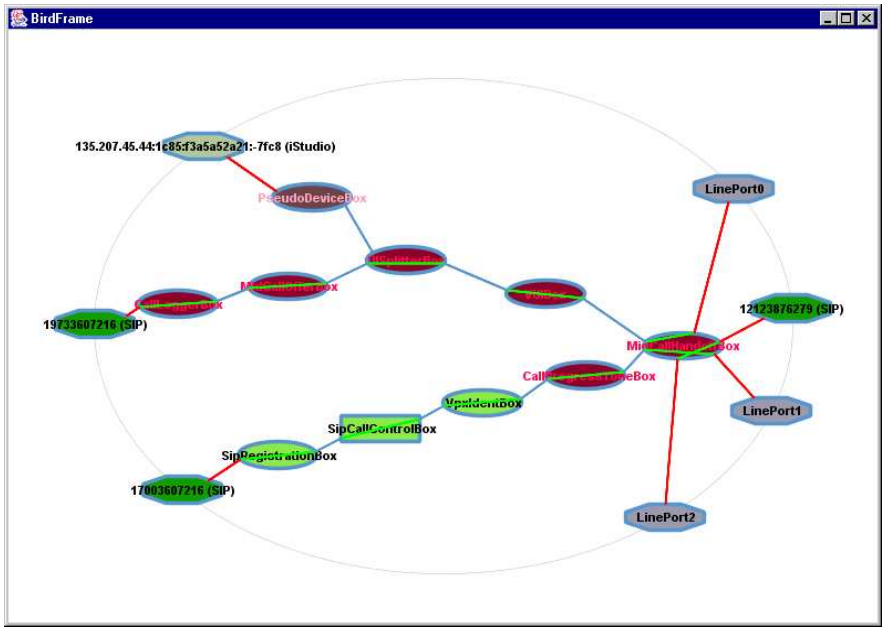
SUGIYAMA, K., TAGAWA, S., AND TODA, M. 1981. Methods for visual understanding of hierarchical systems. *IEEE Trans. Systems, Man and Cybernetics SMC-11*, 2, 109–125.

TAMASSIA, R., AND TOLLIS, I. G. 1991. Representations of graphs on a cylinder. *SIAM J. Disc. Math. 4*, 1 (Feb.), 139–149.

TOLLIS, I., AND XIA, C. 1995. Drawing telecommunications networks. In *Proc. Symp. Graph Drawing GD'94*, R. Tamassia and I. Tollis, Eds., vol. 894 of *Lecture Notes in Computer Science*, 206–217.

ZAVE, P., AND ET AL., C. R., 2002. The Building Box Project. `www.research.att.com/projects/buildingbox`, `www.research.att.com/projects/eclipse`.





Building Box Visualization Service