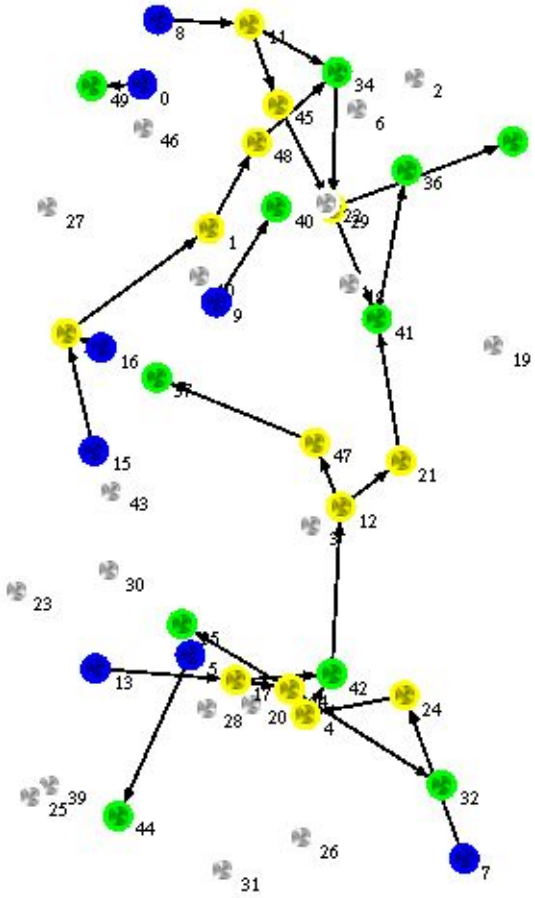
Screenshot and Movie Capture Functionality in iNSpect

by
Matt Gimlin, Neal Erickson, and Charlie Drolshagen



0. Project Abstract

The Toilers research group is an on-campus organization working primarily in the field of ad-hoc wireless networks. Many of their studies are done using a wireless network simulator called NS-2. They have developed a tool called iNSpect (interactive NS-2 protocol and environment confirmation tool) to help them visualize simulations. iNSpect reads from a mobility file (generated after a simulation is run) and then creates a visual display of the simulation during its entire run. This graphical output is done using the OpenGL libraries. Nodes are color-coded according to their current state (routing, transmitting, etc), and the visualization can be navigated via time controls that allow the user to go anywhere within the duration of the simulation. These features allow a researcher to verify the accuracy of node topology, validate new versions of NS-2 itself, and analyze the resultant trace files. However, iNSpect has no capability to record its visual output for demonstration and comparison. Thus, our goal is to add both an image capture and a movie recording capability to iNSpect, while introducing the least amount of additional dependencies.

1. Introduction

iNSpect, the interactive NS-2 protocol and environment confirmation tool, written by the Toilers research group, is a tool used to create visualizations of wireless networking simulation data. The iNSpect program provides an easy, flexible way to view a visual representation of a simulation in real-time. The program has become popular at over forty of the universities that it has been shared with, and so the creators are constantly looking for ways to make iNSpect more useful.

iNSpect runs on a variety of platforms, but it is easiest to get it working on Linux. A user who wants to give a presentation about their wireless network simulation in Keynote or Powerpoint using iNSpect, then, is faced with several choices:

1) Port iNSpect to Windows/Mac
2) Use Linux's screen capture function to get crude screenshots, crop them in another image editor, and perhaps encode them to movies with a third program.
3) Bring two computers to the presentation.

None of these options are desirable. Porting the software to another platform is time-consuming and requires advanced knowledge that all users may not have. Option 2, using Linux's built-in screen capture, is clunky and inconvenient. The user is required to take a picture of the full screen, crop the area he wishes to keep, and re-save the image – this can involve multiple other programs besides iNSpect. For obvious reasons, option 3 is not desirable either.

The goal of this project was to add a fourth option to our hypothetical situation above:

4) Take screenshots and generate movies directly from within iNSpect.

So, our task was to add code to iNSpect that would allow the user to create images and movies of the simulation visuals, keeping the images and movies as cross-

platform and web-ready as possible, and maximizing image and movie quality while minimizing size.

2. Project Requirements

The technical requirements of the project were as follows:

1) Functional Requirements:

a) New tab menus must be added to the iNSpect control window for taking screenshots and movie generation.

b) Within each menu, the user must be allowed to choose the simulation time to take the screenshot or the interval to generate the movie from. The user must also be allowed to specify the number of frames that will comprise the movie.

c) iNSpect's code must have new methods written for generating movies and taking screenshots. The methods will either create a screen capture or generate a movie based on the user's selections within iNSpect.

d) New documentation for iNSpect in the form of a user manual and a programmer's manual must be written. The programmer's manual should be essentially an API for iNSpect, while the user's manual should include detailed instructions for installing, running, and operating the program.

e) The image and movie formats chosen must be common, accessible formats that are readable across all platforms.

2) Non-Functional Requirements:

a) The functional requirements must be implemented before the conclusion of field session 2005.

b) The code written must build upon the existing standards of efficiency and documentation set by the Toilers team.

c) The code must be stable and complete.

3. Design Approach

iNSpect is organized into a distinct hierarchy of classes. The classes that have been modified during this project are toolkitWindow and vizProperties. This project as a whole is an extension of an existing program, and thus the framework of the program is already in place, making our choices as to the design of the program rather limited (see fig. 1 for a class hierarchy).

The class toolkitWindow is entirely devoted to creation and maintenance of the GUI, using the functionality of GTK+. Within it are the functions to create the “widgets” (graphic objects) that comprise the UI, and the functions to connect actions of the user to responses within the code itself. Since much of the new code was written to expand the interface, the majority of coding was within this class. As the code we added was simply extending the functionality of the interface, it was simply appended to the appropriate area of the class.

VizProperties, on the other hand, is a large and complex class, used for several functions. It contains information about the simulation as it runs, keeps track of options from a configuration file, and

controls aspects of the visualization. We chose to add the image and movie recording functionality here – it allowed us to easily access crucial functions for setting up image capture. It also seemed appropriate, as the class that controls the image properties should also be the class that records the images.

Having the coding split between these two classes was also convenient, as it allowed us to work separately, making sure that the interface between our classes was carefully maintained. As much of the code was independent, this easily done.

4. Implementation Details

The final method by which we chose to implement screen capture and movie generation for iNSpect is largely similar to our original design, though not completely identical. Instead of adding an additional library to accomplish image conversion, we have instead used the built-in functionality of GTK+, the library used to create the interface, to save screenshots in formats other than .ppm. This has diminished our goals for adding functionality to the program, in that we are now significantly more limited in our choices of image formats and options for manipulating those formats, but as our goal initially was merely to implement . png image capturing, and we have, our project is not suffering from unmet requirements.

Another change we were forced to make is to implement movie capture via an external encoder, called from within the program. The process of learning a video editing library in C++ was too complex a task to complete within the time we were given, and our clients told us to accomplish video creation by an alternate method. Therefore, we are bundling mencoder, a free video encoding program, with iNSpect and calling it from within the program using fork() and execv ().

The biggest changes we have made to the program are immediately visible. iNSpect now has three tabs in its control window: simulation control, image capture, and movie generation. The image capture tab allows the user to select a format for image capture with a drop-down menu, move the simulation to any point via a slider bar, and capture an image by simply pressing a button. The image name is automatically generated using the system date and the precise simulation time, so duplicate image names are a virtual impossibility, and the user need not worry about typing in the image name each time they wish to save.

Each button, slider bar, combo box, toggle button, etc. in the program is a “widget” created by the GTK library. A widget is a graphic object that is designed to perform a specific function. Buttons are clickable, notebooks hold notebook pages, notebook pages, hold buttons, and so on.

The functionality of the widgets is decided by signals and callback functions. Whenever a widget in GTK is modified in some way (e.g., by pressing a button or by sliding a slider bar), the program generates a signal, which can be thought of as an “event”, in object-oriented programming terms. If this signal is linked to a callback function, then the program executes the callback function. If not, the signal is ignored. For example, because the “Save Image” button's “clicked” signal is linked to the screenCap function in our program, the screenCap function is called whenever that button is pressed. Signals and callbacks are at the very core of the

functionality we added to iNSpect, and are undoubtedly one of the most fundamental tools for development under GTK.

The image capture function that is called when the button in the program's control window is pressed does several things. First, it calculates the total size of an array needed to store all of the image information. Then, it assigns an array of unsigned chars to hold the pixel data. The function calls glReadPixels, an OpenGL function which takes all of the pixel data from the OpenGL buffer and stores it in the array. glReadPixels goes line-by-line, from the lower-left corner of the simulation window to the upper-right (the bounds are calculated automatically each time the function is called), and stores the pixel data it reads into the array specified. The image capture function then writes the data in the array to disk, after first writing the header file for the .ppm image format. If the user has selected .ppm as the format they wish to save the image in, the function stops at this point. If the user selects .png, the program loads the .ppm image from disk in the GTK+ gdkPixBuf class, and then re-saves it as a .png of the correct name before deleting the .ppm file that must be created as an intermediate. On a related note, writing our own .png saving routine would not have been viable - .png is a complicated image format that uses sophisticated lossless compression. Not only would learning how to use this compression have taken longer than the time we had available to us, but using the built-in functionality that was already included in one of the libraries iNSpect uses is a superior solution for obvious reasons.

The movie creation tab in iNSpect has three slider bars and a button to activate movie creation, as well as a check box. The check box allows the user to specify whether or not they would like to save the intermediate .png images that are created prior to movie generation. The slider bars control the movie's start and end points, as well as the current simulation time.

When the user clicks the movie creation button, the function that is called, again, does several things. First, it positions the current simulation time at the beginning of the selected movie creation interval. The screen capture function is called a certain number of times each unit of simulation time, based on the user's selection in the movie capture tab. After the simulation reaches the end of the selected simulation interval, the program stops capturing images and encodes the movie. The way this is done is by a platform-independent process spawn. The process spawned is mencoder, a free movie encoding program. The file that is encoded is, by default, a 25-fps MJPEG encoded file. This is not the only format that mencoder can create by any means, but it is the most reasonable default until other options are available to the user - and they will be added later.

This implementation is not what the programming team or the client had originally envisioned. The reality is, however, that as we actually researched the problem at hand and decided on a method of implementation, we realized that in six weeks there was no good way to implement the solution exactly as the client requested it. He realized this as well, and the design of the program was modified accordingly. The abstract goals of the project - to implement movie and image creation in iNSpect – have been accomplished. The technical details of those goals are not as was originally

planned, but are nonetheless sufficient for both the client and the programmers.

The final implementation leaves the team with many ideas for future improvement of iNSpect. The current limited image formats could be expanded greatly, and the user interface should probably be upgraded to allow the programmer greater control over the format, frames per second, and quality of the movie that is generated. A change that is planned for the next large iteration of iNSpect development is to add movie creation functionality to iNSpect itself so that spawning a new process for movie creation is no longer required. In addition, the Toilers group would like to see things such as zoom control and configuration file manipulation within iNSpect as well.

5. Conclusion

The project was an excellent experience for each member of the field session team. Not only did we gain a large amount of coding experience, but we gained valuable knowledge of how our future workplaces will probably operate.

The project started with some design ideas that were revised as the work session went on due to time constraints and out-of-scope problems. Initially the idea was that all of the movie generation would be done “in-house” by the iNSpect program, without the spawning of any additional processes. Unfortunately, all movie generation libraries available to the team are extremely complicated and poorly documented. Eventually it was decided that adding one of them to the program was too much to accomplish during field session, and so the decision was made to go with an external encoding program.

Also, the choice of image formats available to the user had to be scaled down significantly from what the coding team originally wanted to offer. The reason for this is that we made the switch from an image editing library, DevIL, to using the built-in GTK image saving/loading functions. GTK doesn't save correctly to any image formats other than .png and .jpg as it stands, so we were forced to scale back the options available to the user.

In the future, iNSpect should probably be modified to allow more image formats to be saved so that a greater range of potential audiences can be reached. In addition, most people familiar with the project would like movie creation to be done by the iNSpect program itself, so this will probably be added at some time in the future as well. iNSpect is a project that has come a long way already, and there is a great deal more that can be done with it.

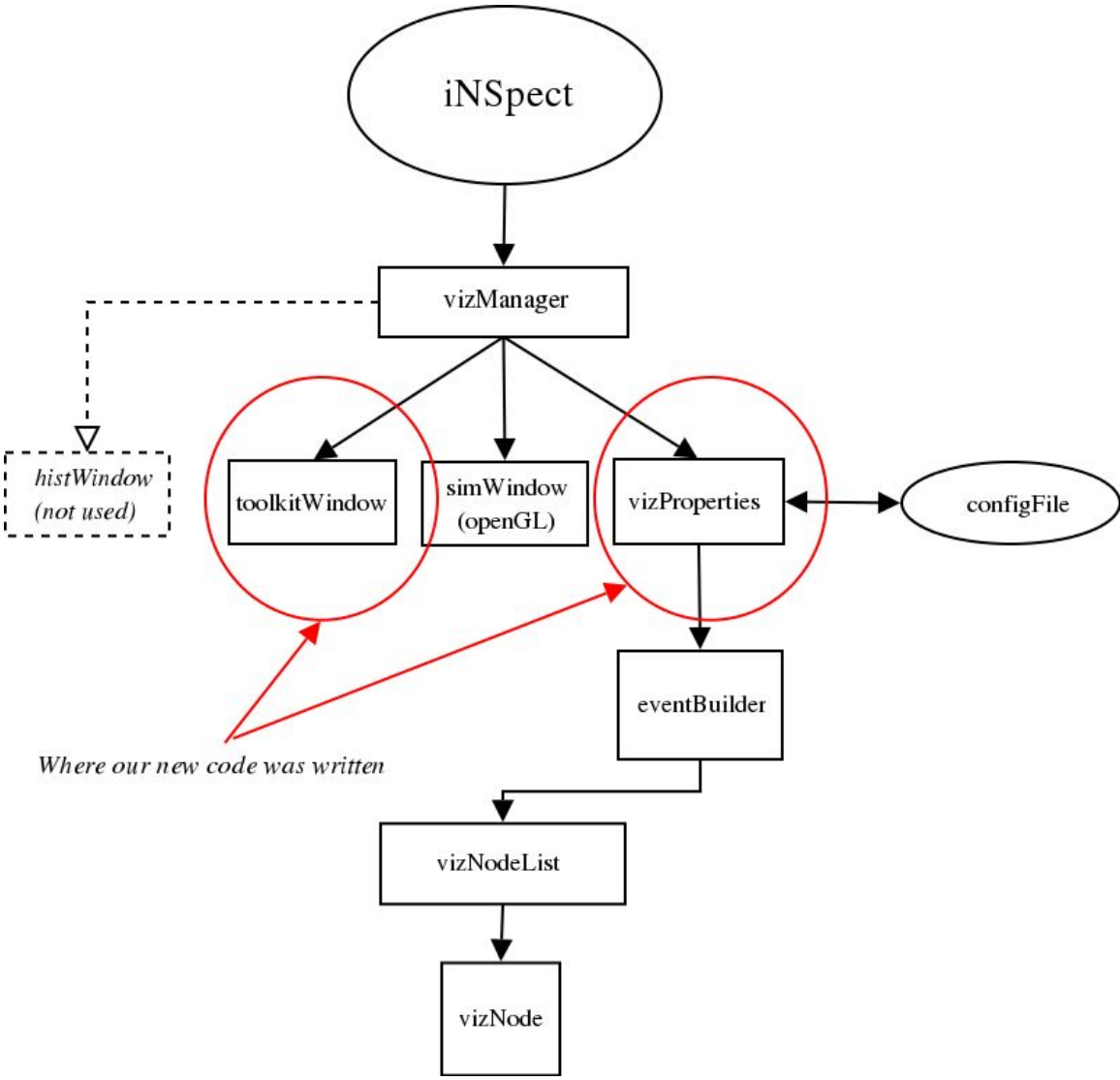


fig. 1 – iNSpect class hierarchy

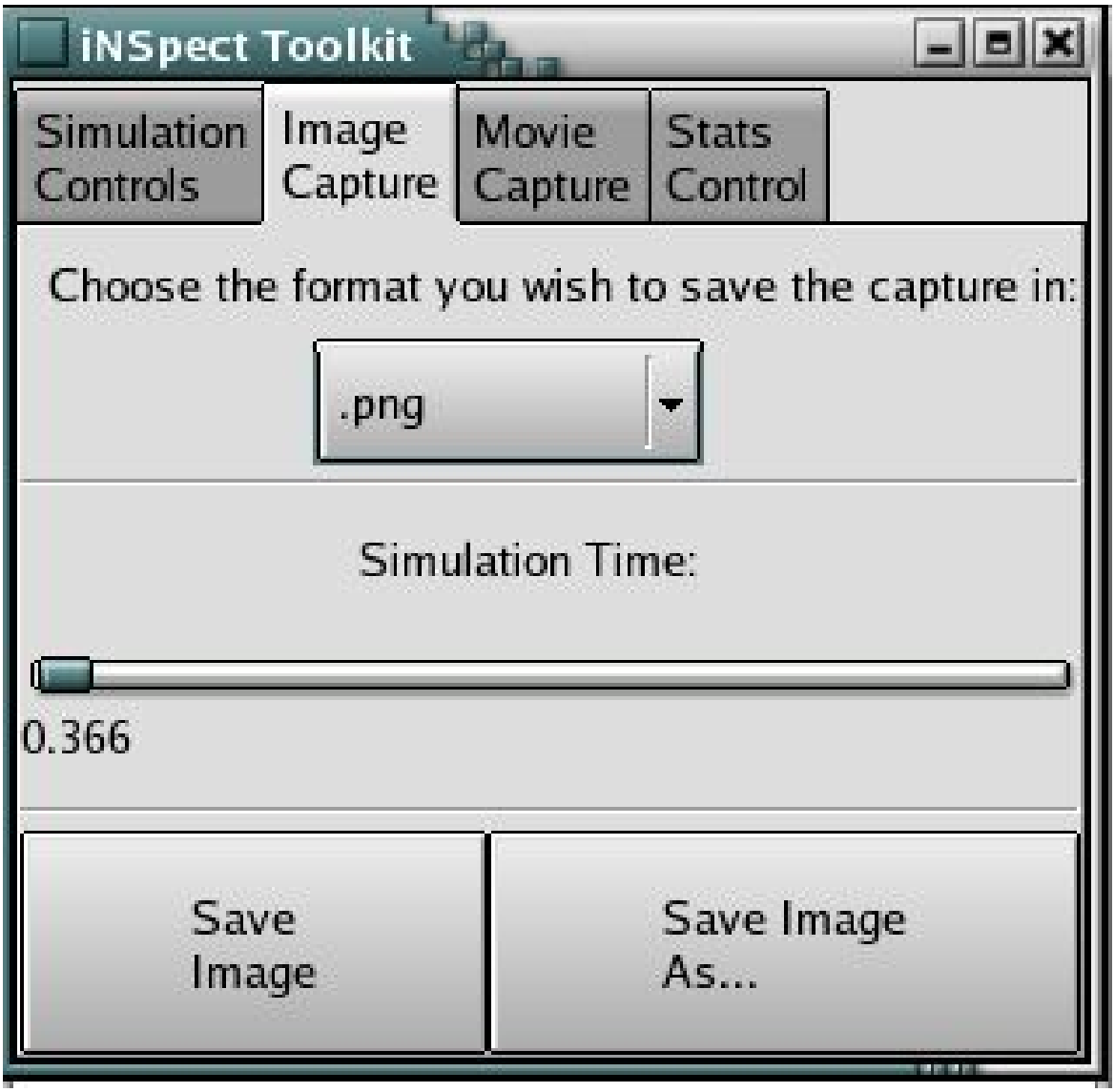


fig. 2 – Modified UI