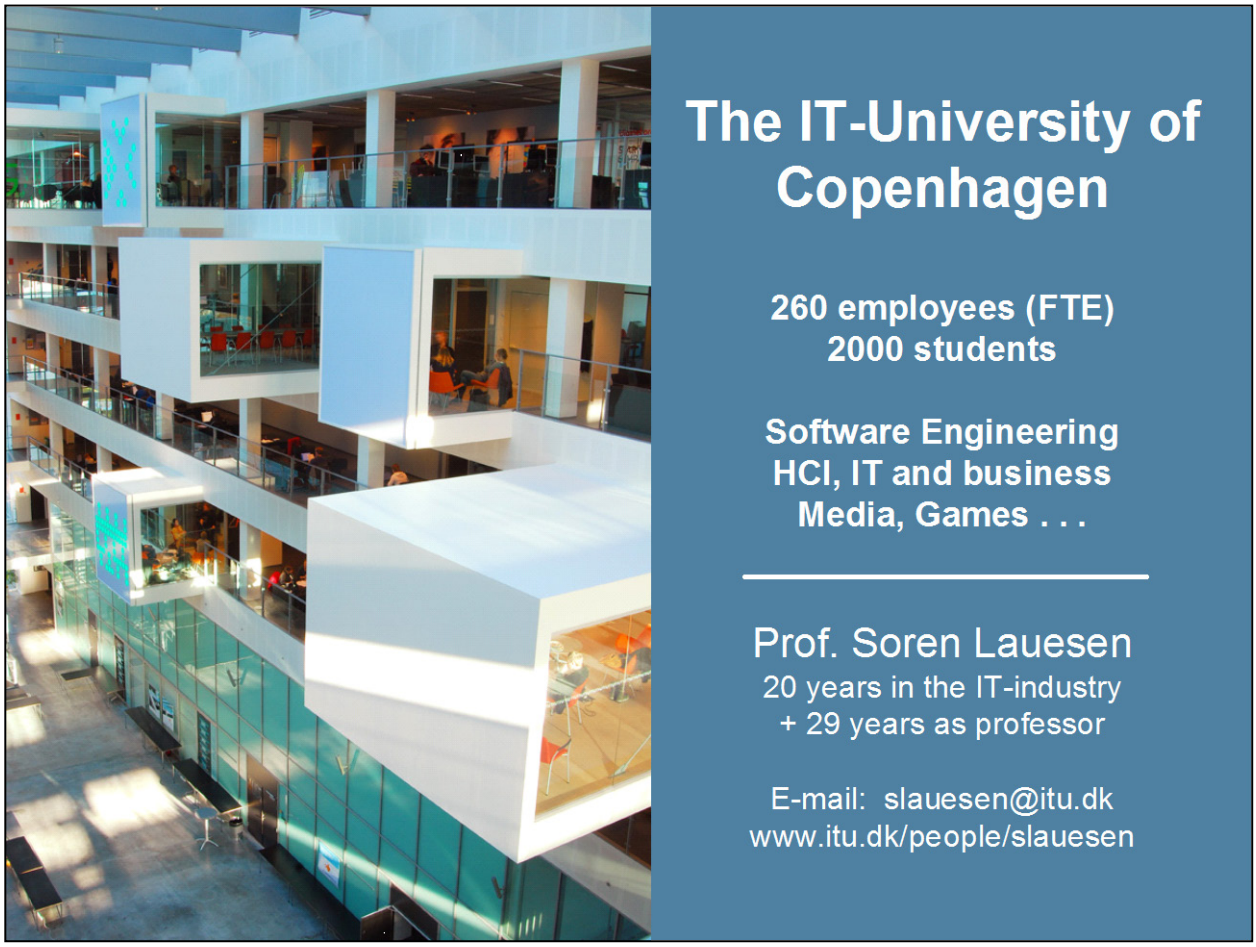
Customizable Data Visualization with uVis
Example: Electronic Health Record

Søren Lauesen, IT-University of Copenhagen, October 2013

slauesen@itu.dk



Contents

1. Health record visualization: The problem

Electronic health records (EHR) are important for improving efficiency and quality in health care, but currently there are many problems with the computer screens (the user interface):

1. They provide poor overview of patient data for clinicians (physicians and nurses).
2. Each medical specialty has its own needs and they change over time (surgery and psychiatry need different screens although they get data from the same database).
3. It is too costly to let the supplier meet these needs.

It is estimated that clinicians on average waste more than an hour a day just to get overview of patient data. The poor user interface is also a source of clinical mistakes, even deaths [2].

Figure 1 shows screens from two present health record systems. The lab results are shown as text only. It is hard to see the essence (result 0) and it is hard to check that it belongs to the patient we deal with. The medication screen shows all medicine orders for a patient. There are many details, but it is hard to get real overview, for instance to answer the questions *which medications does the patient get right now* and *what did he get earlier?*

Clinicians would save much time and avoid many mistakes with screens like Figure 2. The Lifeline screen shows the patient's notes, diagnoses and medicine on a time scale. Icon shapes and color indicate the kind of note, the height of boxes indicate the medicine dose, etc. You can see at a glance which medicines the patient gets now, how long the patient has got them, and how they time-wise relate to diagnoses and notes. The data exist already in the database; we just show them in a different way. No health record system uses such screens today.

One way to deal with the problem is to ask the system provider to deliver *data visualization* screens such as Figure 2 for each medical specialty. However, this is very expensive.

The obvious solution is that local IT staff develop new screens in cooperation with local users. Surprisingly, this is not possible with present tools [10]. Tools that can show data as in Figure 2, require programming beyond local staff capabilities [7].

Present tools for non-programmers cannot show data as in Figure 2. Imagine that a local developer tried to use a drawing tool (e.g. PowerPoint) to show a Lifeline. He would draw circles, boxes, etc. on the screen and manually define their position, color and other *properties*. However, he couldn't make the system extract data from the database and automatically generate the circles, boxes, etc. and their position and color. Real programming is needed.

2. The solution: A drawing tool where properties can be formulas

Uvis is like a drawing program, but any property can be a formula somewhat similar to a spreadsheet formula. As an example, the formula for the horizontal position of a diagnosis combines data from the existing database with data from the time scale to compute the position. Spreadsheet formulas are within local developer capabilities and are not "real programming". The data visualizations in Figure 2 were made with simple uVis formulas in a few hours. With a bit of training, local IT staff in a hospital could have made them. No programming was needed.

Formulas can also specify how the system must respond when the user clicks or types something. In this way local IT staff can make real applications that use existing databases.

Uvis is not only for health records. It can be used for any system with a database.

Currently we look for partners that can help bringing the tool into industry.

Figure 1. A traditional Lab-result screen and a "modern" medication screen

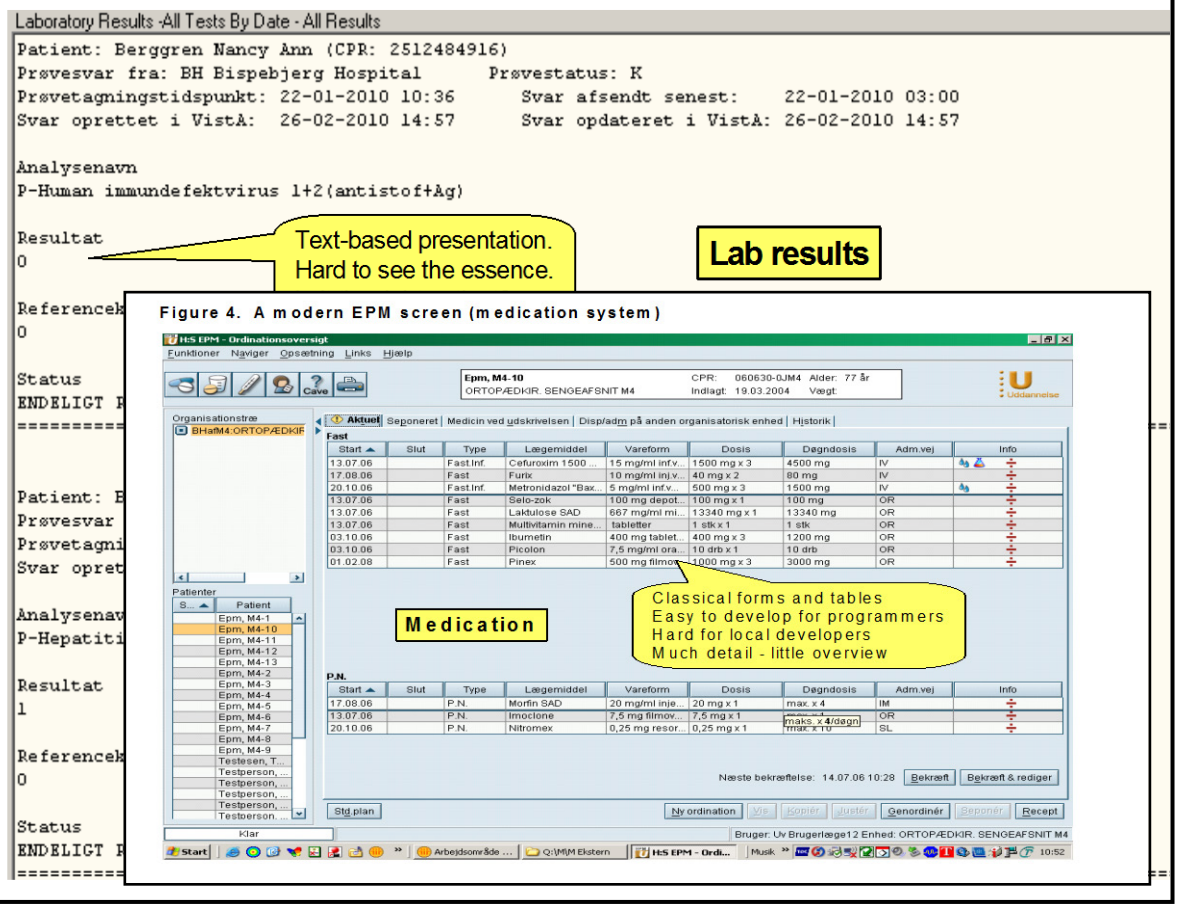
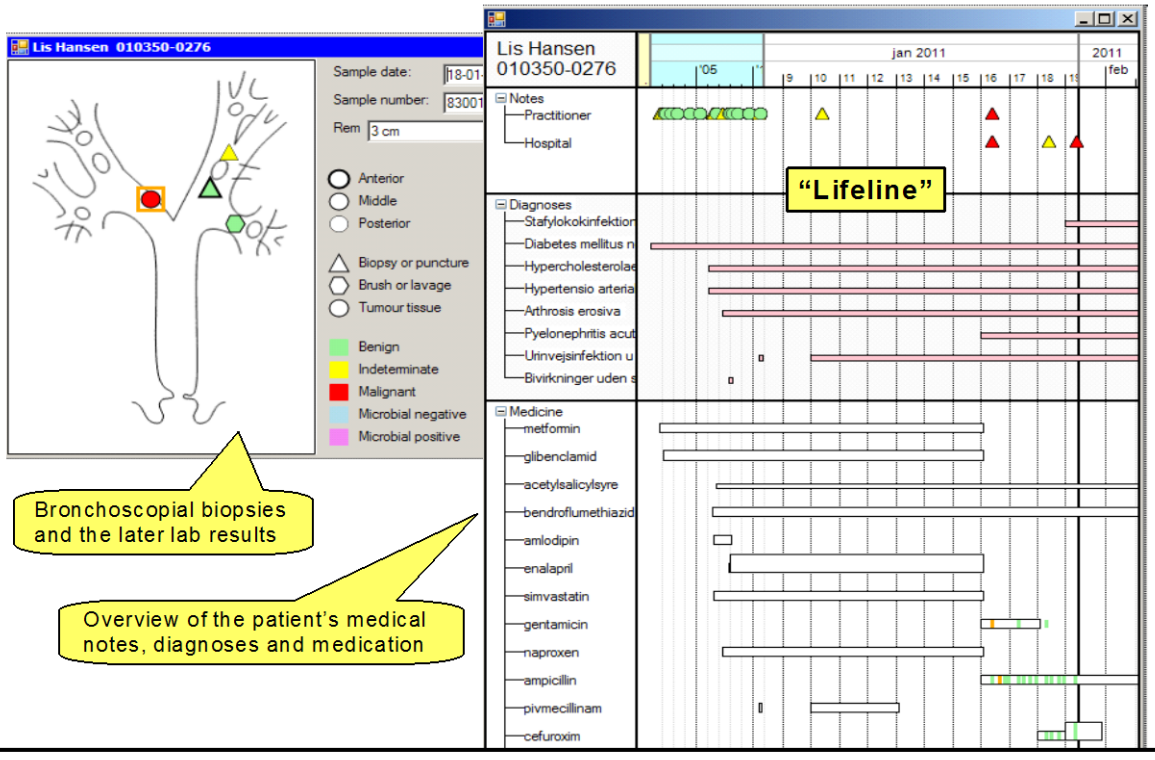


Figure 2. Two screens with data visualization – made in 4 + 6 hours with uVis – no programming



3. Data visualization tools

Data visualization means showing data as color, position and shape, rather than as text and decimal numbers. When we perceive data as text and decimal numbers, the brain uses the language centre and thus blocks many other activities. For instance, it is hard to listen to speech while making sense of Figure 1. Data visualization such as Figure 2 doesn't block the language centre.

In general, data visualization enhances cognition by revealing patterns, trends and outliers, and allows users to make better decisions [1][3][11].

Data visualization has become a large research area in the last decade. Researchers invent new ways to show data graphically and they test how users perceive the data. The visualization is in focus and researchers care little about integrating visualizations with intensive production applications such as medical records.

In principle, the system provider could deliver data visualization. However, this is very expensive, particularly if each department wants its own screens and they also have to be part of normal system maintenance. In addition it is hard to get the screens right and easy to use in the first attempt. It requires experiments with real users, which adds further to the cost.

Amazingly, developing visualizations such as the Lifeline is hard even to professional programmers. As an example, a small, very competent software house provides visualization for a broadcast company's planning. Their screen is similar to the diagnosis part of the Lifeline screen. It had taken them half a year to develop it, and they were surprised to see how it could have been made in a day with a mature version of uVis.

Some developers question that a new tool is necessary. You can get ready-made packages for graphical presentation such as pie-charts and bar diagrams. And you can get tools for making your own data visualization. What is missing? The short answer is that these tools fail in one or more of these ways:

1. The tool is hard to integrate with daily production applications.
2. It requires too much programming.
3. It provides only predefined visualizations.
4. It provides only predefined interactions with the end-user.

4. How the local developer will work with uVis

Uvis means *unified data visualization*. Details of uVis are published in [4][5][6] and compared with other tools.

In this section we show how an IT-interested physician or nurse (a local developer) would use uVis to construct the screens in Figure 2 and deploy them in the department.

Connecting to the database

First the local developer needs a *data-map* file that gives uVis access to the existing database. Most likely he will use an anonymized test version of the database. The central IT department would give him the necessary permissions and help him set up the data-map file.

The local developer opens the uVis tool and tells it to select the data-map file. His computer screen will now look like Figure 4. The tool shows an empty user screen that will become for instance the bronchoscopy screen. In the bottom right panel the tool shows the database tables and their relationships. The developer can open the tables to explore the actual data. Our experience is that local developers do this a lot to understand what the data really are.

Constructing the screens

The local developer will now drag components from the toolbox into the user screen in the same way as with many other tools. He will define properties for the components with the property grid. As an example, he will drag an icon component into the screen and specify its FileName property as "bronchia.gif". The "bronchia.gif" file contains a standardized diagram of the bronchia and the diagram will now appear on the screen. This is similar to existing industrial tools.

However, the icons that show the biopsies as circles, triangles, etc. are different. The local developer dragged a Glyph component to the screen and specified its properties as formulas that made the Glyph component create copies of itself and give them the right shape, color and screen position according to data in the database. We will explain this in some detail.

Figure 5 shows uVis when the local developer has constructed most of the bronchoscopy screen. He sees the bronchoscopy screen exactly as it will look to the end-user. He can also interact with it as the end-user would do. He has selected one of the icons that represent a sample (e.g. a biopsy). He sees its properties in the property grid. He has also opened the table of bronchial samples. The selected icon corresponds to the first row of the table.

The property grid shows that the sample icon is a Glyph component. Depending on data, a Glyph can appear as different shapes: circle, triangle, etc. The Left position of the icon is computed by the formula *x-7*, where x comes from the database. For this specific icon, the formula gave 106 as the result and uVis put the icon 106 pixels from the left border of the bronchoscopy screen.

The shape of the Glyph is computed by the Type property. The formula retrieves the *kind* field from the database and chooses the corresponding shape: triangle, hexagon or circle. While the developer types the formula, uVis shows what is possible to type right now (called Autocomplete or Intellisense). When the developer has typed the formula or changed it, the system updates the user screen immediately.

A formula may refer to formulas in other components, which again may refer to other formulas. Just as in spreadsheets, circular references may occur. Uvis shows this and other errors in the error list panel and as colored marks in the formulas.

The developer has specified the connection between the icons and the table rows with this Glyph property:

Rows: Form -< Bronchial

It works this way: *Form* is the bronchial screen. It is already connected to a patient data row so that it can show the patient name and patient ID at the top of the screen. The -< operator symbolizes a walk along a crow's foot in the graphical data map from one record to many. In this case uVis walks from the patient row to all bronchial rows connected to the patient. Then it generates a Glyph for each bronchial row. As a result, the end-user will see one icon for each of the patient’s biopsies.

Figure 4. Starting uVis

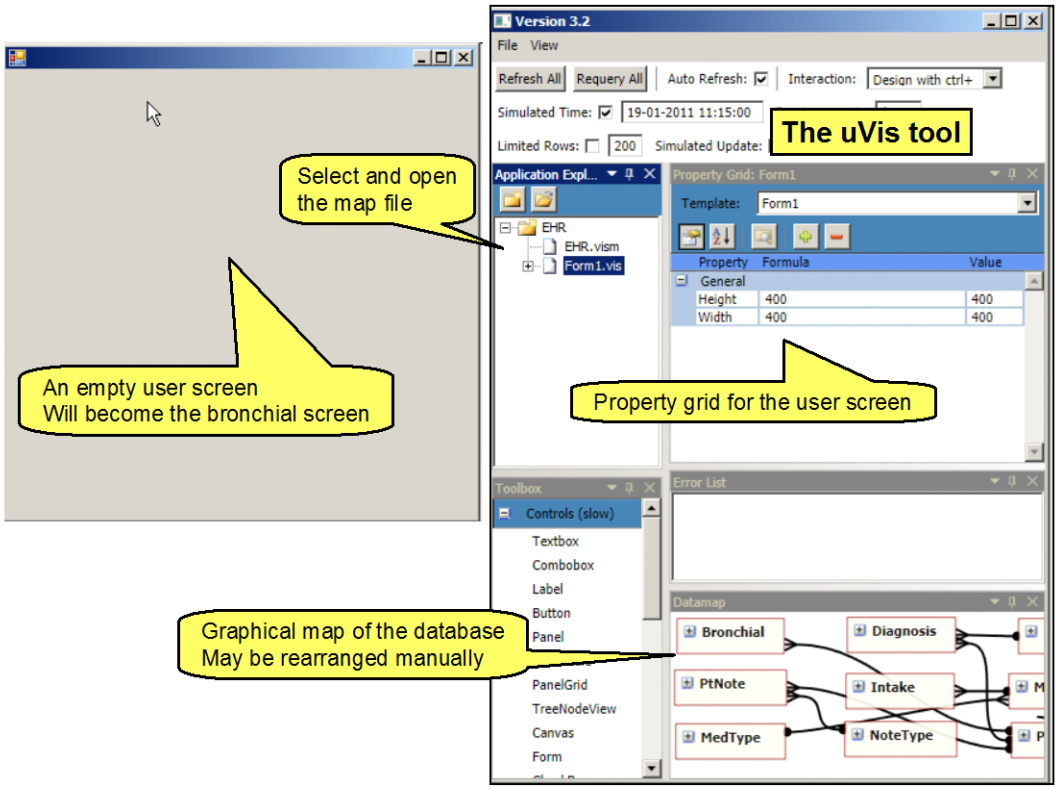
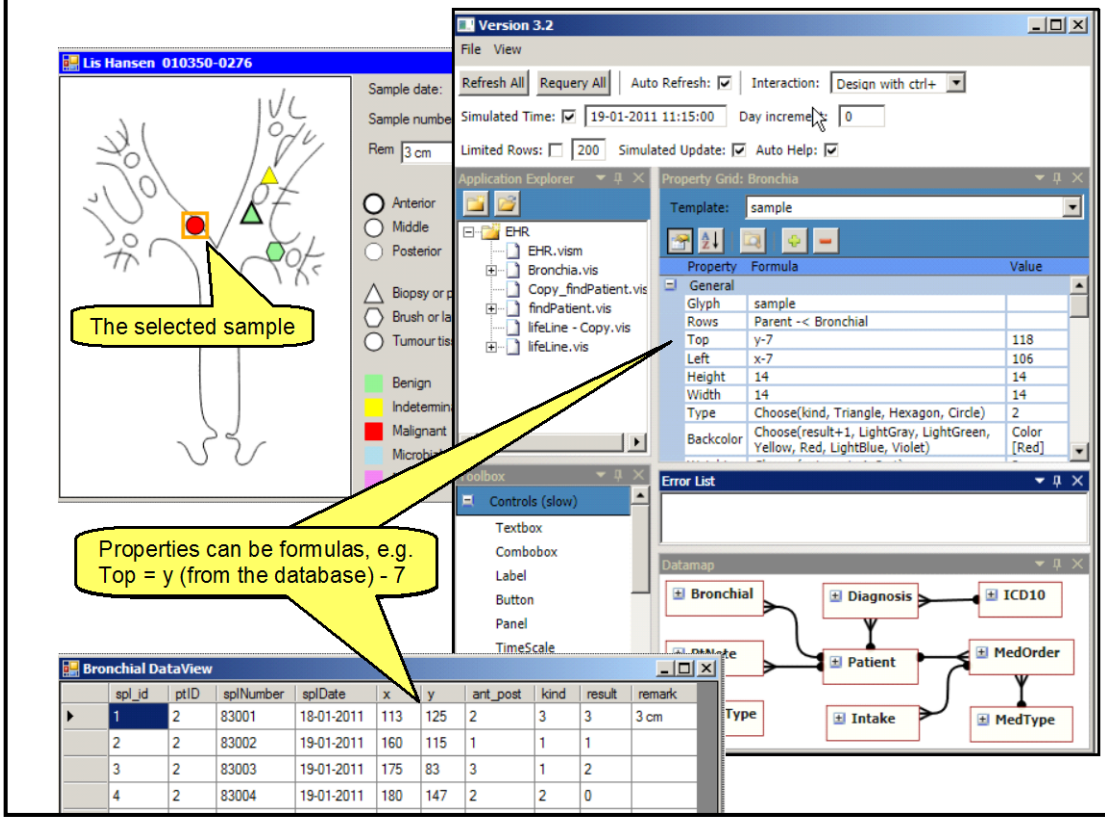


Figure 5. During development – properties are computed by formulas



Form -< Bronchial is a novel, compact notation. If the developer had to express it in the usual way as an SQL statement, it would look like this:

```
SELECT Bronchial.ptID, Bronchial.y, Bronchial.x, Bronchial.kind, Bronchial.result, Bronchial.ant_post,
Bronchial.splDate, Bronchial.splNumber, Bronchial.remark
FROM Bronchial WHERE [Bronchial.ptID] = 0103500276
```

Furthermore, he would have to make a program that inserts the actual patient ID (0103500276) into the SQL statement and sends it to the database.

The -< operators can be combined. There is also a >- operator. It too walks along a crows foot in the graphical datamap, but from the many-side to the one-side. In the Lifeline screen, you can see each medicine intake at the hospital (the small colored bars at the bottom of Figure 2). This is achieved with this formula for the intake bar:

Rows: Form -< MedOrder -< Intake

When the local developer has constructed a screen, the tool saves it as a file (a *.vis* file). It is a simple text file that can be edited with Notepad or in a graphical way by means of the uVis tool.

Testing the screens

The local developer has made the basic test of the screens while he constructed them. Whenever he typed or changed a formula, the system immediately retrieved data from the database and showed it exactly as the end-user would see it.

However, the local developer should make additional tests of the screens. He should discuss the screens with the local users to ensure that data is shown correctly and he should test that the screens are understandable. He should test that the screens show abnormal data correctly; preferably he should have a small test database with abnormal data for this. And he should test that the queries don't overload the database. Here he needs a large test database for measuring performance, for instance an anonymized version of the production database.

He might also have local users play with the system. This is easy. He puts the map-file and the vis-files in a folder on the department's file server. The local users can now open the map file in the same way as they would open a Word file. As a result, uVis will connect to the test database and open the start screen that the local developer made. Then the user interacts with the system exactly as he would in real use.The user doesn't see the uVis tool and cannot change the screens. In our example, the developer would make a start screen where the end-user can select a patient and then open the patient's lifeline, bronchial screen, etc.

We have paid much attention to the test situations. For instance, it is easy to switch between databases. You make a copy of the data-map file for each database and modify the database connection string. When you run a test, you simply open the proper map file. You can run uVis in various test modes, for instance simulate that the date and time is something that matches the data you test with. You can avoid that uVis makes real database updates, but only simulates that data has been updated. To avoid that a database query by mistake takes a very long time, you can limit database access so that the system never retrieves more than for instance 200 rows from a table.

Let the clinical staff use the screens

After testing, the local developer has to get the necessary rights to access the production database. He also sets up a map file for production (it needs to connect to the production

database rather than the test database). Further, the end-users need rights to open the map file and access the production database.

As an example of daily use, imagine what the clinical staff would do during bronchoscopy. While the surgeon carries out the bronchoscopy, the nurse opens the map file, selects the patient and opens the bronchoscopy screen. When the surgeon takes samples, the nurse marks the spots where the sample is taken. They select a shape to indicate the kind of biopsy and enter the sample number. The screen stores this in the database according to what the local developer has specified in the formulas.

When the clinicians later get the lab results, they record them on the bronchoscopy screen in this way: They select the sample icon and change the color of the icon by clicking one of the square color buttons. At a certain point, they use the screen to decide how to treat the patient.

When in the future the system receives the lab results in digital form and records them in the database, the results automatically appear on the bronchoscopy screen as colors.

You can develop screens with uVis for any application with a database that is accessible to another program. We have used uVis to make visualizations of many kinds of data (e.g. hotel room assignment, quality indicators for software, and hits on web pages) in many traditional forms (e.g. bar charts, curves, spirals and pie charts).

Close integration with existing applications
In the scenario above, end-users opened a map-file to see the existing data through uVis screens. Another approach is that an existing application asks uVis to open the map-file and show the uVis screens. This requires a small change to the existing application so that it calls uVis when the user for instance clicks a certain button on an existing screen.

Since uVis can boost the work of professional developers, they might use it themselves in existing applications.

5. State of uVis

The uVis principle seems easy, yet it has been hard to realize. It was particularly hard to combine the formula principle with relational database principles in a smooth way. This *object-relational impedance mismatch* [8] has been attacked by many researchers with modest success.

Uvis is quite stable and performs well. It can for instance retrieve 10,000 data rows and show them as 10,000 glyphs in less than a second. It shows the Lifeline screen in the example in 0.7 seconds, including 0.4 seconds to make 8 queries to the database. (Measured on an ordinary 2 GHz PC with 2 GB memory and a local MS Access database.)

We have tested the ease-of-use with the kind of developers we aim at. They are non-programmers, but have IT expertise corresponding to some spreadsheet and database experience. We found that uVis is as difficult as spreadsheets. Most of the non-programmers learn the basics in 2 to 3 hours, but need more experimentation on their own. We have also tried uVis with programmers. They learn it faster. Some of them had programmed visualizations professionally and said that uVis would have saved 90% of their time.

The main weakness of uVis is that it currently is based on Windows Forms. We would like to develop a web-based version and make it accessible also from iPhones and other mobile devices.

Currently we look for partners that can help bringing the tool into industry.

6. Competences

Uvis is developed at the IT-University of Copenhagen (ITU). ITU was founded in 1999. It has around 260 employees, 2000 students, and graduates around 250 students a year. It covers research and teaching in several areas of IT. Of particular relevance for this project is the research in software development, CSCW, ERP systems, user interface design, mobile systems, and large software acquisitions. ITU has a couple of research projects in collaboration with Microsoft, and others in collaboration with hospitals.

Team members

Soren Lauesen, project manager, full professor, has worked 20 years in the IT industry as a developer, project manager and department manager. Half of the time has been in multi-national companies. He has worked 29 years as a professor at the Copenhagen Business School and ITU. His research has mostly been done in cooperation with industry or public organizations (including hospitals).

Lauesen is widely recognized for his approach to systematic user interface design, and for his approach to requirements in public acquisitions. His textbooks in the area sell around 1500 copies a year; he has helped around 30 universities worldwide to teach according to the books; and he has helped around 30 companies to use the methods. More than 30 people a day (excluding search robots) visit his web site to download documents such as tutorials, slides for the books or research publications. See http: //www.itu.dk/people/slauesen/

Soren Lippert is a former heart surgeon who for nearly two decades has worked with improvement and standardization of EHR systems. He joined the uVis project in 2008. One of his roles is to be a "local developer" without programming skills, but with good knowledge of spreadsheets and simple databases.

Mohammad Kuhail, Ph.D student with a British Master's degree in software engineering. He is a creative and careful developer who invented amazing ways of using uVis for advanced visualization, boosting system performance, etc.

Kostas Pantazos, Ph.D student with a Swedish Master's degree in software engineering. He is an experienced and creative developer, and a data-mining expert with experience from a large Danish bank. He anonymized a large Danish EHR database (300,000 patients) as part of creating test data for uVis.

7. References

[1] Card, S. K., Mackinlay J. D., Shneiderman, B. (editors). Readings in information visualization: using vision to think. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[2] Hellebek A, Skjoet P. (2010) Aggregated review of root cause analyses related to medication. Proceedings in Information Technology in Health Care: Socio-Technical Approaches.

[3] Keim, D. A., Kohlhammer, J., Ellis, G., Mansmann, F. (editors). Mastering The Information Age - Solving Problems with Visual Analytics. Eurographics, November 2010.

[4] Kuhail, M. & Lauesen, S.: Customizable Visualizations with Formula-Linked Building Blocks. In IVAPP 2012 proceedings, February 2012.

[5] Kuhail, M., Pantazos, K., Lauesen, S.: Customizable Time-Oriented Visualizations. In ISVC 2012 proceedings, July 2012.

[6] Lauesen's web site: http://www.itu.dk/people/slauesen/
The site contains a description of uVis with detailed examples as envisioned in April 2009. At that time the tool was called VisTool.

[7] Myers, B., Hudson, S. E., Pausch, R.: Past, present and future of user interface software tools. ACM Transaction on Computer-Human Interaction, Vol. 7, No. 1, March 2000, pp. 3-28.

[8] Object-relational impedance mismatch, Wikipedia. March 2012. http://en.wikipedia.org/wiki/Object-relational_impedance_mismatch

[9] Pantazos, K., Lauesen, S., Lippert, S. (2011): De-identifying an EHR database - Anonymity, Correctness and Readability of the Medical Record. Proceedings of MIE2011.

[10] Pantazos, K., Lauesen, S. (2012): "Constructing Visualizations with InfoVis Tools - An Evaluation From A User Perspective", In Proceedings of the International Conference on Information Visualization Theory and Applications.

[11] Plaisant, C., Mushlin, R., Snyder, A., Li, J., Heller, D., Shneiderman, B., and Colorado, K. P. (1998). Lifelines: Using visualization to enhance navigation and analysis of patient records. In Proceedings of the 1998 American Medical Informatics Association, Annual Fall Symposium, pages 76–80.