Selected projects of Audrius Meskauskas

High throughput analysis projects

My oldest projects I would like to tell about are research works, implementing bulk statistical comparison between actual shape development and development, predicted by various mathematical models. Experimenter photographs the developing object (like growing mushroom how it raises the cap after turning into horizontal position). My task was to process the scanned photos and video tapes, numerically comparing the observed shape with the shape as predicted at the given time by the various mathematical models. These models were expressed as systems of partial differential equations. It was my task to implement numeric solutions and automatically tune model parameters for every experiment. We performed both statistical tests on how well various models fit the data and how statistically model parameters are distributed. We (also me) proposed several new models and also first time have made a clear proof that some popular past models do not match experimental data. From these works we have published three publications in *New Phytologist.*

The project started as "user friendly", trying to use Matlab, Mathcad and Maple. After wasting enough time with these tools I have concluded that they just do not do that we need. I have switched completely into C++ in that all project was finally implemented and all published results were obtained. The project used flat text files to store all data it processed. At the end we also wrote the visualizer that demonstrates how changing parameters impacts the predicted shape, also in C (while I was asked to rewrite visualizer as Java applet for publishing as a supplementary material on a web).

Later I used similar approach to implement and compare mathematical models of industrial fermentation, even reusing some code. It was actually a simpler task as the experimenter provided a number of automatically recorded curves, how do the various measured parameters change over time in many repetitions.

Sight, web robot generator

With the raise of bioinformatics, a lot of researcher groups have started web servers, providing one or another type of automated analysis of the user supplied data. Frequently it was interesting to compbile multiple web services into pipelines but the sites have little support for this: they were sites for human beings with web forms and HTML output. Even worse, the design of the sites was changing over times, so any automated tool with complex workflow was short living: it was necessary to build it quickly, launch like a rocket for a time and collect the results; a week after one or more services maybe will not run as they were. In Ulm university I have developed and later used a user friendly tool for the fast creating of web robots and then connecting them into workflows.

Sight is a user friendly application, implemented in Java using Swing. In various specialised cases it uses some external libraries like Velocity template engine, Kopi Java compiler, Weka statistical engine and the like. Sight, however, does not use any kind of web robot generation framework nor any kind of workflow framework - at the time of programming there was no any really suitable.

Building the web robot starts from scrapping the page where the web form is located. The operator can see which forms and which parameters (hidden parameters including) are present and provide the sample values. Then the sample input is submitted. The submission code supported both GET and POST and was able to follow HTTP, HTML and JavaScript - based redirects.

On the next step, the parser for the server output is generated. As the sites were very diverse, I have developed a number of parsers:

- The table based parser locates values in the tables that were commonly used to format

the output. It is able to select one of several tables by order or below/above some signature text, also step into nested tables.

- The Stalker parser uses the Stalker algorithm to generate regular expressions surrounding values that the user must select. The Stalker algorithm simulates the evolution process. In case if the algorithm cannot detect the signatures reliabley, they can be manually entered.
- The simpler and actually more useful version of the parser allowed to mark the noticed signatures directly on the server output text with mouse. Human operator may be smarter in understanding which parts of response and can serve as a good anchors.
- The XSLT parser was written for these rare servers that have been returning XML; it uses the operator-provided transform to alter the near arbitrary XML into internally standard intermediate form that can be easily parsed. The development GUI that I have written for this parser allows to debug the proposed transform, seeing how it processes the provided server output.
- One more very simple parser expects the plain text table between the string signatures.
- Later I also added WDSL agent that was also able to send WDSL compliant requests.

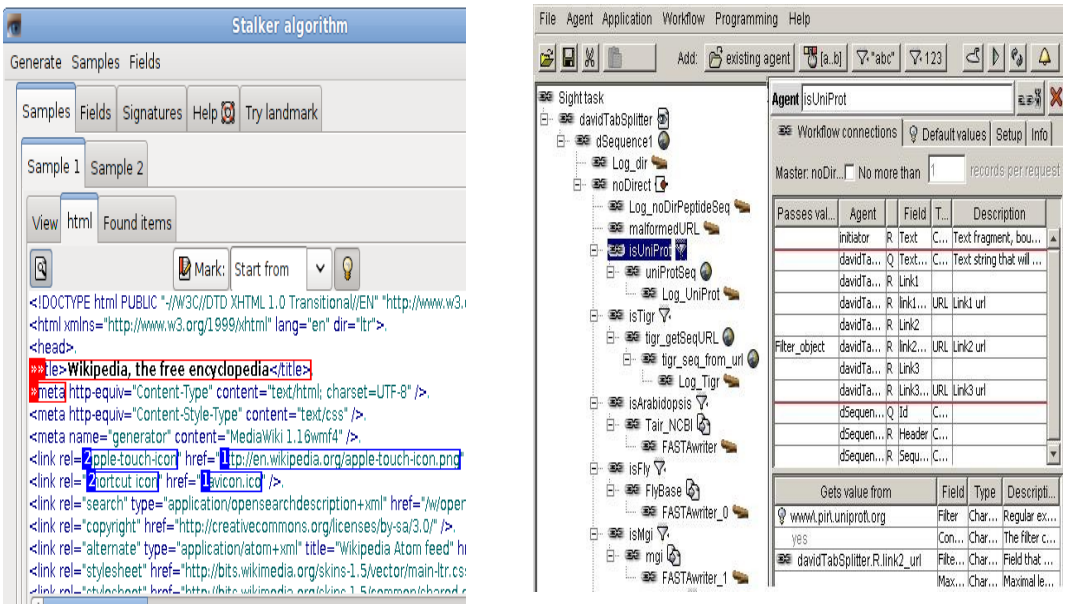


Fig 1. One of the web agent generators (left) and the workflow builder (right).

After both sender and parser are configured, the tool uses Apache Velocity template engine to generate a source code of the new robot/agent. The generated .java class implements all submission, parsing and can return the data description structures containing names and types of the request and response. One request usually results many responses like multiple search hits. After that, the class is compiled with Kopi Java compiler (Sun's compiler was proprietary at that time) and dynamically loaded.

Finally, I also wrote user friendly tool to connect these agents into workflow. To build the workflow, it is necessary to know which named input parameter of the slave agent must be connected to which named output of the master agent. The flow also contains some specialized agents that provide initial input and write the output. Similarly as agent, the workflow was also generated with template engine and compiled, after that it was ready to run.

Sight flows are much faster than the work of human operator. Not just all data passing is automated; every agent runs in a separate thread and has its own input and output queue, so all involved web servers (some quite slow) work in parallel, providing kind of distributed computing. The

flow can even run for some time if one of the servers is transiently out; data in the queues may still provide enough work for other nodes.

We implemented 60 second pauses between requests, automatically increasing them if the server takes long to respond. To avoid sending the same request, all responses were cached on a disk. The system was likely civilized enough as we have never received any complaints from webmasters.

We published one research article on the system itself and later another on various interesting results we obtained using it.

GNU Classpath

At the time of me joining, GNU Classpath was a top priority FSF project to provide a Free alternative of the proprietary Java implementation from Sun Microsystems. Java was popular; Java was attracting people and becoming the base of more and more FOSS projects, making them dependent on the good will of the single company. The project was also a good place to meed top level developers from commercial companies (Red Hat, Aicas) and various interesting research people who needed a Free Java for various projects devoted to the virtual machine (the first JRE offering a JIT compiler was CACAO, a reasearch JRE running GNU Classpath).

Because of the certification reasons, it was important to have complete implementation of the system libraries. I have picked org.omg and javax.swing.text.html.parser where nobody was working seriously.

I have used javax.swing.text.html.parser in web agent generator so this API was already familiar to me. It is a SGML-driven library; you need to start from writing a helper parser for a provided SGML document. The parser must maintain a stack of the current HTML tags (nested into each other where appropriate) and close them automatically. For instance, <tr><td><td></tr> is a perfectly valid HTML and the parser must fire two </td> events despite these two tags are not present in the input. The parser was covered by the numerous automatic tests.

CORBA is actually a clear and logical communication library; just the original OMG specification is written in a quite difficult style. I have implemented the basic GIOP protocol, the old style ORB, the Portable Object Adapter, locator and activator support, RMI over IIOP and the naming service. CORBA has some funny features inside; for instance bytes are aligned to allow easy structure loading - this only adds overhead in Java as this language has no built-in structure type. The protocol officially allows both Big and Little endian but Sun's Java only understood Big endian messages. I added tansparent Little endian support to my implementation - be GNU Classpath at least somewhere better than Sun's SE. The finished library was able to communicate with Sun's CORBA implementation without any issues.

SourceForge contains the COST project (CORBA Open Source Testing) with numerous CORBA tests that were contributed by major providers like IONA in the past. I used this test suite to check how well my implementation actually performs; it has passed all tests that Sun's code passed, and it also passed some more.

I also fixed some old and complex bugs in GNU Classpath JTable and JTree, making these classes fully functional and ready to use, also contributed a number of tiny fixes and extensions over all system library.

GNU Classpath was a school which converted the former programming researcher into professional developer. This is where I have learn about CHANGELOG, patches, code repository, Ant that we used to build run tests, GNU build tools (GNU Classpath also contains some C code), need to comment the code, need to cover the code with tests and so on. The overall discipline inside the GNU Classpath project was very strict; currently I know that many even commercial projects are developed much more loosely. Also, this project gave good understanding about many dark corners of Java system library.

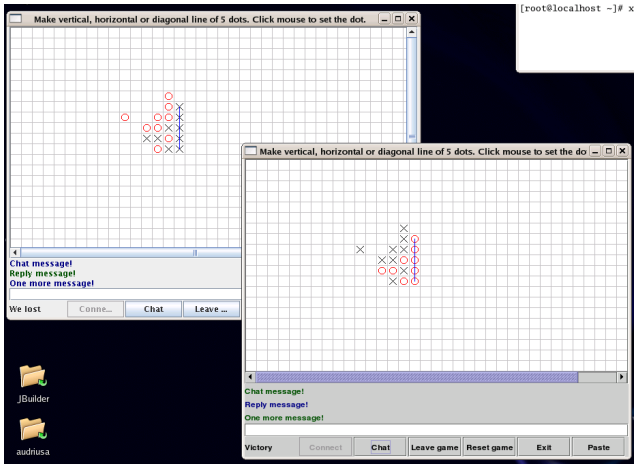


Fig 2. GNU Classpath (top) talks with Sun's Java (bottom) via CORBA platform, playing a simple distributed computer game. I have also tested this game between the two machines running Classpath under Linux and Sun's side under Windows.

Three dimensional models of morphogenesis

This project was one of the first attempts to simulate three dimensional (rather than two dimensional) development of the fungal structures. The analyzed mathematical models assumed that all involved cells (hyphal threads) are actually identical, and the form emerges from chaos just through the identical algorithm of behavior than runs in every cell. This our hypothesis was later reviewed in *Nature*, the top level journal for biology, writing a dedicated article and citing our team as a first people who have proposed this hypothesis. I was responsible for all programming, simulation and data fitting part, also contributed to the development of mathematical models. Hypha interact through some abstract (in real world likely electric) fields that they can both generate and sense; they can respond by changing the growth rate and direction. I implemented multiple versions of both scalar and vector fields with various properties. Again, the major task (that converts computer game into scientific project) is statistical fitting of the experimental data into mathematical models, estimating the parameter distribution and overall goodness of fit.

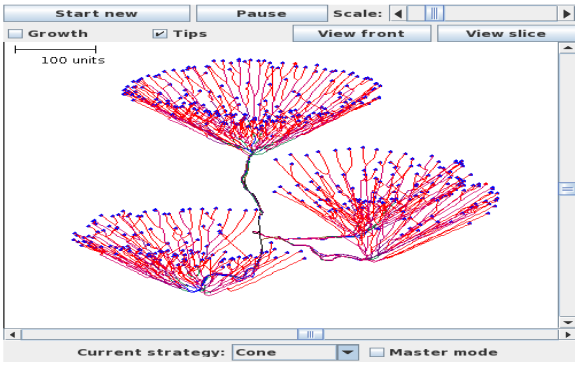


Fig 3. A “proof of concept” simulator that we used to obtain the grant (the serious analysis requires bath mode).

While I also wrote a “live” visualizer that did a good job when obtaining grant application, the real simulations of three dimensional development are computational intensive and were done in bath mode, using Manchester supercomputing facilities and parallelised algorithms. The program was generating frequent XML snapshots of the current “universe”. These XML files contained enough information to resume the simulation using any of them as starting point. Later another tool converted

these files to the series of images and finally into AVI animations.

Rio Viz

My first task in Spectraseis was to implement a specialized map-based visualization of the newly discovered seismic attributes. While Spectraseis has evaluated several map frameworks in the past, the required level of interaction was so complex that a specialized application was required. The data were stored partially on a PostgreSQL database and partially on a shared filesystem. It was a requirement to provide user-friendly tuning on how far from the point interpolation is still good enough to display the data. The data distribution was quite noisy and the user needed the dialog to pick the range of interest from the graph of statistical distribution of values. I have received the detailed specification about that the system should do but not how.

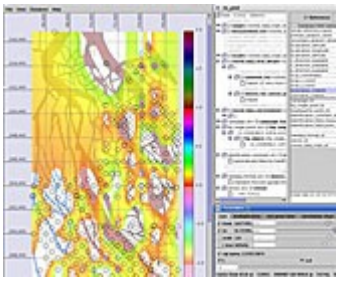


Fig 4. RioViz displays the complex and partially transparent shape of computed data (shades of orange) over the existing map.

I have picked the existing interpolation library in C++ and the open source tool to draw charts and histograms (JFreeChart). I have implemented the rest myself in Java, fully matching and in many cases exceeding the requirements.

Rio Range

After completing the first task I started to look around and noticed that data processing specialists spend a lot of they time on a highly inconvenient Matlab application to show various charts and spectrograms of the recorded seismic data and to mark manually areas of interest (in time axis) that are too noisy for the further analysis.

The main problem with Matlab tool was the lack of multithreading. At the same time due huge amounts of data to process the application had many slow steps. As a result, the Matlab tool was freezing for many seconds if not minutes after each user action. It was not possible to make more input before completions of the currently running heavyweight threads.

Using Java, I was able to move heavy calculations to the background, utilizing multiple CPU cores, keeping application responsive and maintaining possibility to cancel unneeded processes if the context changes after the new user input.

RioRange uses database server (PostgreSQL, JDBC) to keep the output of the whole team in one place and allowing to share the results. At the same time it performs spectral decomposition of the data from files on a shared filesystem (native call to FFT algorithm) and presents multiple parallel spectrogram charts, where the noisy areas can be marked with the mouse. After marking action the program starts preparing the updated data in the background but transparently cancels this if the user thinks the situation is clear enough and marks more regions.

This application has raised the productivity by the good order of magnitude and (as it was my

own project from idea through design till product) likely earned me the status of the team leader in Spectraseis. The company started to get serious interest in software development and hired two additional programmers for me. I have prepared several clearly formulated tasks on extending RioRange with new functionality. These tasks were successfully implemented by my team that now I needed to teach quickly about test driven development, iterations, releases, code repository, Bugzilla and so on, remembering the rules of GNU Classpath. RioRange and RioViz initially had only several users but needed to work reliably, as bugs in data processing were very costly for use. To satisfy this requirement we wrote a number of JUnit tests, including tests for GUI.

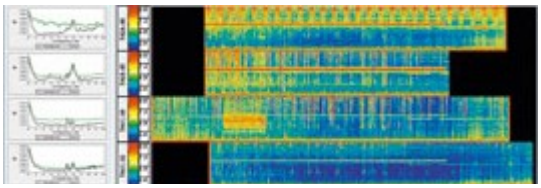


Fig 5. RioRange. The operator marks valid-invalid areas in the spectrograms on the right. After each marking, all view must be recalculated but the operator frequently wants to mark again without waiting for.

My tasks for the team were usually quite technical, like “I need a class with this and this functionality, here is the algorithm and talk with this and this geophysicist for additional details on it. Your class must implement the interface I give you. Please include JUnit tests to be sure about this and this”. I was continuing active development myself as well. We were working next table to geophysicists, talking with them during various meetings so it was not difficult to figure out which new features or even new tools are required.

RioImport

As Spectraseis uses a database to store the data it acquires on a field, it needs a separate tool to add the field data to that database. The database at the end has evolved into sophisticated system with many connected tables for multiple levels of hierarchy (survey, measurement, measurement section, measurement channel, pictures, additional data, etc). Even worse, due company profile the data sets had different structure for near every survey, and the tool needed to be adapted quickly – frequently in days only and without the time for proper testing. The company initially used Matlab scripts that were struggling dramatically, making another major bottleneck in the data processing. I started a challenging project to develop importation tool that would be flexible enough to import almost “every data into every database schema”, as long as the talk is about seismic measurements. The tool runs numerous tests on the consistency of the incoming data. Simple tests (like if the number is between the two boundaries) are defined through GUI. However most of the tests are written in SQL (or PL/PSQL where SQL was not enough), they include cases as “looks like the same device measures in two places at the same time” and the like. Before importation, RioImport shows data and metadata preview for every measurement. A team of geophysicists can work on the same data set; they see each others actions in real time through the CORBA based centralizing server. The data import is transactional.

I have designed all concept and wrote the majority of the code, while some clearly defined components (like a GUI area to show signal traces or data readers for various formats) were delegated for the other members of the team. Again we used some libraries to produce embedded charts but we found no any reasonable “super framework” to build a quality control, data visualization and database populating application of this complexity.

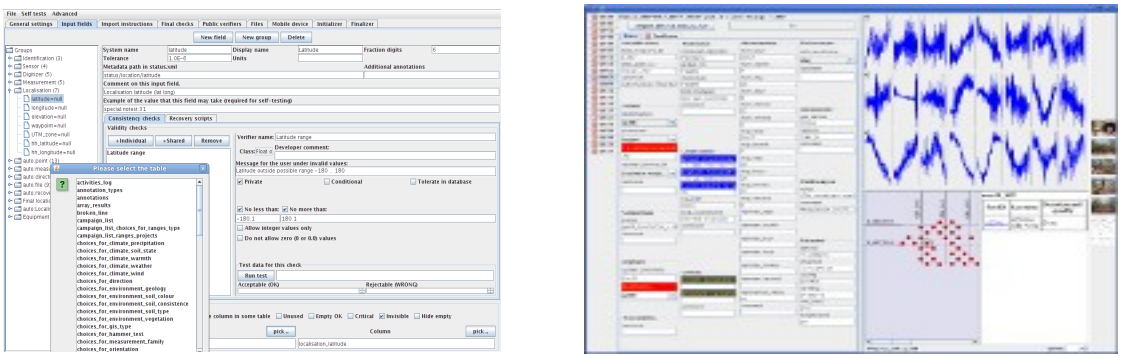
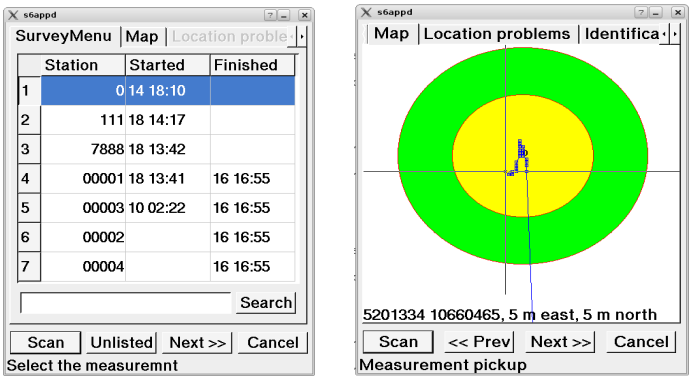


Fig 6. RioImport configurator (left) and the main window, usually seen by geophysicist (right).

The field handheld

On the field during the survey, the technician needs to deploy the measuring equipment but also collect some other data about the surrounding terrain, rocks, vegetation and potential sources of the interfering noise (like roads). He also makes pictures. These actions are taken during every of hundreds of measurements. It was a huge work to sort photos from the cameras and enter the data into computer from paper protocols that were initially used.

Using the TDS Nomad under Qtopia our team gradually developed a field device that has forms to enter the data, picks GPS from the internal chip, takes pictures with internal camera, downloads test data samples through USB and in the office automatically delivers all data to the central storage through wireless. The fields and chapters of the on screen protocol that have been changing all the time can be updated for all accessible devices also through wireless. The user friendly Java tool shows all visible devices on the screen allowing both bulk and individual actions with them. Nomad has ARM processor, but we anyway installed and used PostgreSQL (to access it from C++) and many Linux tools on it. The device also has laser scanner that we also use to scan barcodes on equipment. The barcodes contain destination tag and device places the scanned data into appropriate fields of the protocol by



itself.

Fig 7. Handheld screen shots, obtained while debugging the application on desktop (QT code is cross platform).

Additional difficulty was the need to merge protocol that is taken during the measurement placement with the protocol that is taken during the measurement pickup. Placements and pickups occur in random order and are not necessarily done with the same device.

I had some really excellent developers in the team, and we reused pieces of the old Spectraseis code to download test data samples (written in Perl) but the project would not have been even started without my initiative that originated from the past OpenMoko project. It have done a lot of "proof of concept" scouting (can we use PostgreSQL on device? Can we join Qtopia and Perl? Can we get wireless working the way we want)? After being sure everything works I was delegating the now obvious and doable task to my people, frequently with in general working skeleton-prototype. Some developers just continued with that skeleton when others were reimplementing the production application from scratch – both approaches worked well.

I also wrote two separate Makefile's to build the same code either for Qtopia or for desktop PC. This allowed to debug the applications efficiently without the need of emulators or remote debugging on device.

At that time members of my team also started some more independent projects, me more caring about the successful integration of tools under development and maintenance of the libraries that are shared between the multiple projects. Under the pressure of the library versioning problems (one project requires newest version but another uses the older one and there is no time to upgrade it) we have finally switched into Maven and Nexus repository. These tools make the versioning issues relatively solvable.

Android

While Qtopia devices perform well in the field, it has been highly interesting to test Android as well. This alternative approach includes using the mainstream mobile phone or tablet rather than specialized device. Even high end phone is cheaper than these specialized devices and commonly has much better hardware specifications. Android also provides the highly efficient Google Maps component. Our Android Field Office system has not been yet deployed on the field but it is under development and performs well in pilot tests. Apart Google Maps, it also uses QR codes heavily. Google Maps gives lot of features very easily, it is simple to place custom overlays containing not just points of interest but also complete own drawings on the top of existing maps, using available coordinate converters. However they seem way more efficient with the working Internet connection.

Fig 8. Our data downloader on Google Galaxy Nexus

The most interesting part of this task was the requirement to support the wired RJ45 connector, something you probably will never see on any mobile phone. To support this nice feature, all our Android stuff now must be rooted as I need to send *ifconfig* to the attached USB to RJ45 converter. Easy to say – everything is in the cloud! The data measuring device is not, and it only RJ45 connector, no other.

One of the interesting Android possibilities is that now programs can send crash reports to the cloud, using ACRA library. They also self-update from the cloud using S3.

RioGrande

RioGrande is Spectraseis next generation tool that includes all functionality (and of course a lot of my old code) from RioViz, RioRange and RioImport, as well as many additional features. Unlike

previous projects, we have outsourced the programming of RioGrande to the five people team in St Petersburg, Russia. This team was found and the initial contact was made by Vlatko Davidovski, a member of my team in Spectraseis.

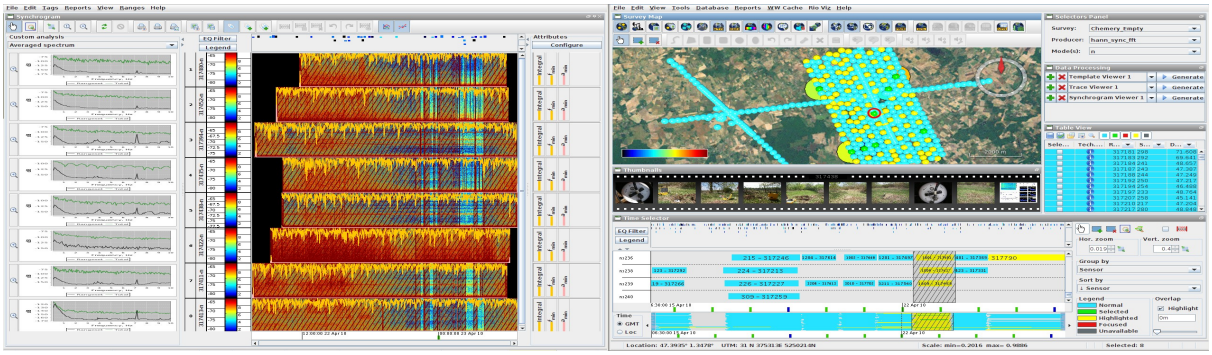


Fig 9. RioGrande (multiple components are OSGI plug-ins).

I have prepared the initial specifications on how the first iteration of RioGrande should look like, described in details all needed functionality for clarity also wrote the unimplemented interfaces for all its components. I also contributed a lot of code from my previous projects, explaining where and how it is possible to reuse it and where needed wrote a simple wrappers the demonstrated where and how the code is working. Me and Vlatko fly to St Petersburg every several months to talk there a lot and make sure there are no any misunderstandings. Russians commit the new to SVN that is closely watched by Vlatko who also builds and checks the development snapshots. When after the first development iteration some bug reports appeared, we started to process them through Bugzilla. All this ensured relatively smooth start without typical problems of outsourcing. In several months we had an impressive new tool. Similarly to Eclipse, it internally uses OSGI framework, so it is possible to develop, deploy and debug plug-ins without even restarting the main application. Switching into plug-in concept also allows to have multiple individual development tasks that do not conflict and can be easily distributed: just everyone develops the assigned plug-in or the next version of it. Finally plug-ins can have independent development and release cycles. With RioGrande I am already more an organizing person than the primary developer, but I keep the code overview, watch the quality of the implementation, write and run tests, do profiling and in some cases need to add features and fix bugs personally, just to prevent talks that one or another task is too difficult or impossible to complete. I am also still personally developing RioImport and in some degree the field device stack.

Schlumberger Petrel

It has been strategically important to integrate our tools with the standard tools, used by various companies in oil industry. One of such tools is Petrel, a Windows/C# application with very powerful visualization capabilities. Our group (with me being the main developer and lead designer) have developed various modules and plug-ins, extending Petrel framework in standard way in order to provide access to our data. In the opposite to various open source tools) these large industrial frameworks are not documented in freely available Internet sources, where as a rule only marketing materials can be found. Normally an expensive purchase (order 100K) is required to get the developer API and attend the necessary courses. Even if we do not plan to use Petrel, knowledge of its internals and architecture may bring a valuable input in making design decisions. Also, it makes the group more aware about the real possibilities and limitations of these monster frameworks.

AppTornado AG

My major tasks in AppTornado AG were around the website this company uses to work with people who advertise with AppTornado system on Android devices. The site has relatively complex backend, combining PostgreSQL (where transactions and consistency are vital, like in money matters) and Cassandra and Redis (where amount of data and ability to scale over multiple machines are more important). The website itself is also surprisingly complex, combining JSP, Apache Velocity, Google Web Toolkit (GWT), some custom JavaScript and extensive CSS styling to deliver very impressive content. You can see some non authenticated pages in AppBrain.com. The build system is based on Apache Ant.

Apart website itself (that I design using pixel to pixel drawings from designer) I also develop maintain its test suite, based on Selenium and PhantomJS. We make automated screen shots for all our numerous pages, for various resolutions and also separately for mobile devices (using Android emulator). This allows to check them easily to see that nothing is broken after we tweak CSS. Daily builds, tests and screenshot series are available from Jenkins continuos integration server.

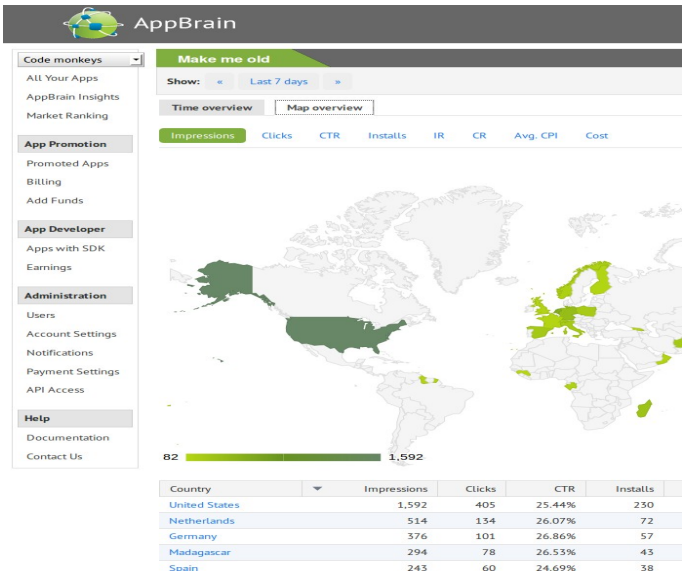


Fig. 11. User data reporting tool, implemented by me using GWT (data come from PostgreSQL).

Among other interesting tasks I has been responsible for implementation of they credit card payment system, based on BrainTree, designing they database schemas, modifying and maintaining the continuos integration build system, RPC services (based on Google Protocol buffers) and the like. The team also has highly advanced development workflow, keeping all code in GitHub and developing new features in branches with frequent merges.

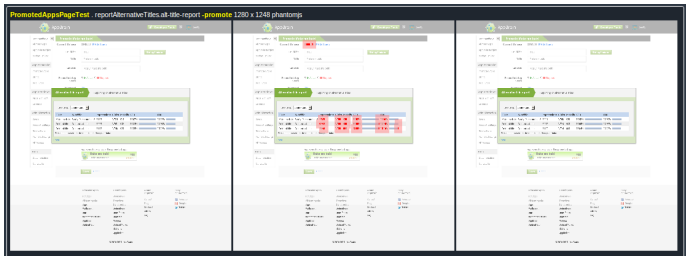


Fig. 12. Differential screen shots automatically produced in continuos integration builds (my Selenium project).

Other

I am familiar with ANSI/AAMI/IEC 62304:2006 standard used to develop mission critical applications.