

Why we chose ZK when creating Sciformation ELN

Introduction

Since 2006, I have worked on so-called Electronic Laboratory Notebook (ELN) software, which is used by scientists to document experimental setups, procedures, observations and results. I am myself a chemist, i.e. an autodidact in the field of programming. My first ELN software, open inventory (OE)¹, was written in PHP and Javascript, being part of my PhD thesis². It is open source software.

The development took place between 2006 and 2010, when asynchronous

The logo for 'open inventory' features the word 'open' in red and 'inventory' in blue. The letter 'o' in 'inventory' is stylized as a hexagonal chemical structure with a red dot in the center. A blue horizontal line is positioned below the text.

communication and single-page web applications came up. To deliver a user experience comparable to desktop applications, I developed a custom basis for CRUD applications where the data is loaded asynchronously through an invisible *iframe*.

OE is still quite popular, but certain design decisions limited the range of potential users to university research groups and small companies, while larger companies or research institutes missed a fine-granular permission management and customization options, for instance to support a broader range of science disciplines. The latter is particularly difficult *and* important, as the future development of science is very difficult to predict.

To overcome the intrinsic limitations of OE, we wanted to develop a new software system together with the renowned Max-Planck-Institut für Kohlenforschung³. OE's data design and many good features served as blueprint for the new development, which we called Sciformation ELN (SE)⁴. We wanted to design a clean basis while being able to quickly make adjustments not foreseeable at the project start, in close collaboration with the users. We wanted to offer an

The logo for 'sciformation ELN' features the word 'sciformation' in blue and 'ELN' in red. The letter 'o' in 'sciformation' is stylized as a hexagonal chemical structure with a red dot in the center. A blue horizontal line is positioned above the text.

extended functionality and – more important – flexibility to quickly add form templates for new types of experiments, without any changes to the core software components.

In an early project phase, we (unsystematically) evaluated multiple cross-browser GUI frameworks for Java, like ZK⁵ or GWT⁶. Our main goals were

- to avoid browser-specific issues and
- to reduce the required workload by using pre-built widgets.

Requirements and evaluation

In our evaluation, the license of the respective framework, the existing user basis and the documentation were the first points to check. Packages with a license containing a strong copyleft like GPL or AGPL⁷ were not an option, as we wanted to create commercial software, so that the development costs could be financed by license sales. A broad user basis was important to reduce the risk of a framework to be abandoned, we wanted to ensure that the software component will be continuously maintained in future years and decades.

To achieve a desktop-like user-experience, the immediate user interaction was a very important point, which we could pre-evaluate using demo installations on the web. JSF and many derivatives⁸ were quickly excluded from further evaluation at this point, as they seemed to require full page reloads for many operations. This may have changed meanwhile, but waiting was not an option for our project.

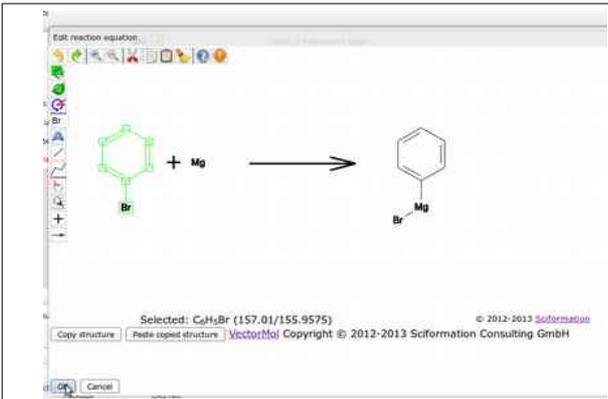
Once a promising candidate was found, the installation of the framework was the following step. GWT was our first serious candidate, probably at version 2.2.0. It was rather complicated to install and took us days to create a very simple GUI, we were soon frustrated. Furthermore, integrating any existing Javascript code fragments seemed very complicated to us.

Therefore, we were close to creating a custom GUI basis again, when my colleague came across ZK. It was easily installed and worked out-of-the-box, offering a large set of well-tested widgets, good page layout help for different screen sizes (“flex”) and a fast user experience. The documentation and demo site were supreme, both in quality and extent, offering different learning approaches for the technology. The integration of legacy code worked well, the technical design seemed very clean and logical. We stopped the evaluation at that point, as we were fully convinced.

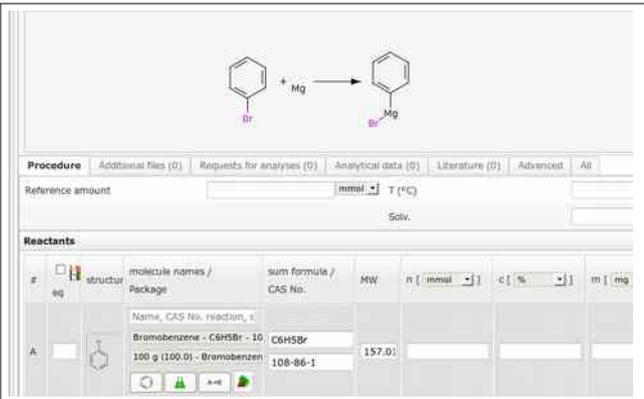
What we like about ZK

At this point of decision, we were not aware of other striking features of ZK, which we step-by-step started to take benefit from:

- the **ZUL form system** offers the flexibility required to quickly design and adapt forms, the Include component allows to nest and reuse components.
- In combination with the **databinding** system (of ZK 5), we achieved the current status, where new sub-forms can be realized without any modification of the backing controller component. As map entries can be bound to GUI properties, we are able to save custom properties as name-value-pairs to the database, offering almost unlimited flexibility to design future document templates without any modification of compiled code. Other frameworks like JSF do offer databinding as well, but are much slower than ZK and do not offer an acceptable user-experience.
- We could easily develop **custom macro components** to edit molecular structures (Scheme 1) or to manage analytical data (Scheme 2), special functionality required for scientists.



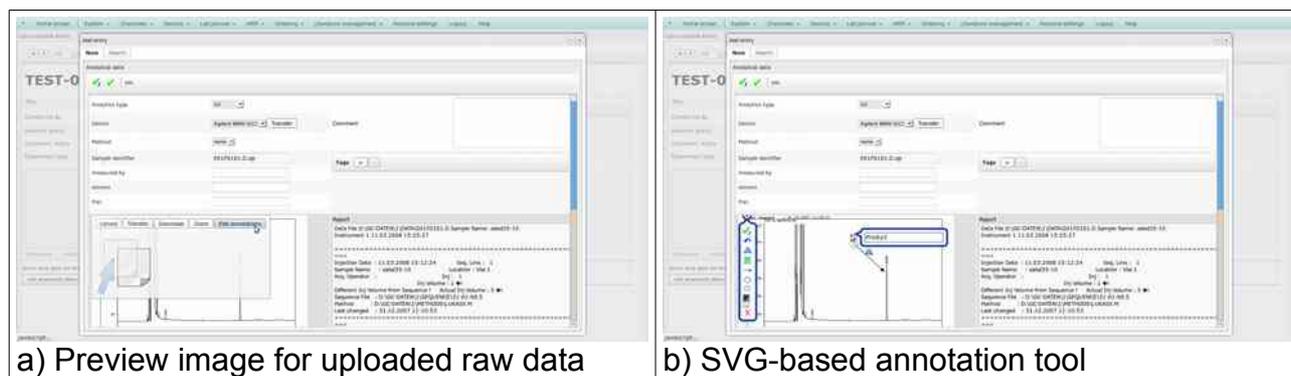
a) Integrating a molecular editor⁹



b) Depicting chemical structure formulas

#	structure	molecule names / Package	sum formula / CAS No.	MW	n [mmol]	c [%]	m [mg]
A		Bromobenzene - C6H5Br - 10	C6H5Br 108-86-1	157.01			

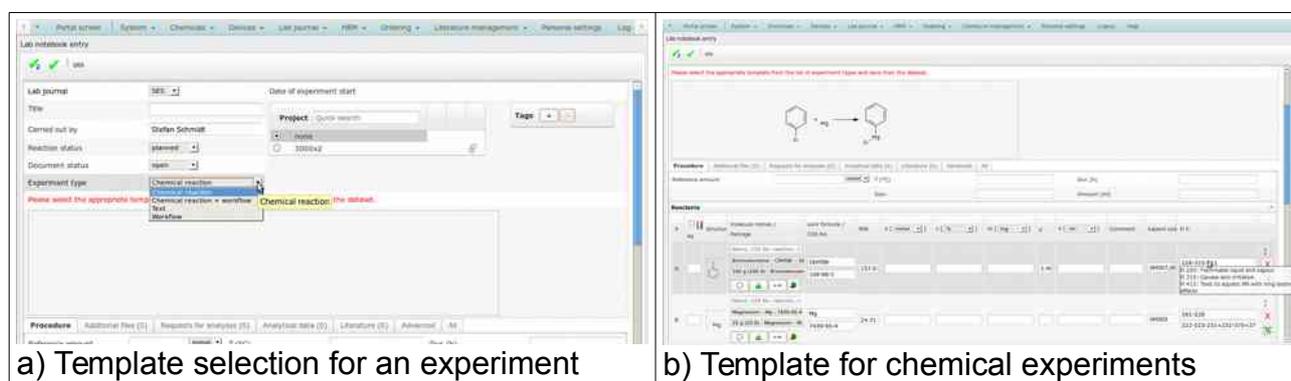
Scheme 1: Chemistry-specific macro components

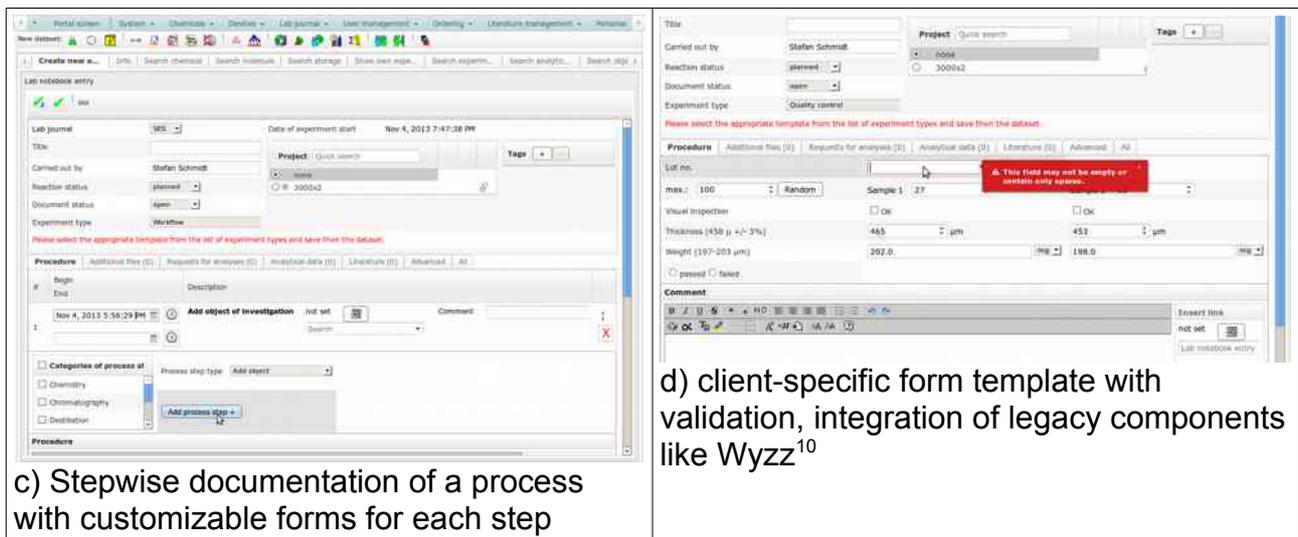


Scheme 2: A custom macro component to handle analytical data

- Moreover it was simple to create custom **derivatives of the included widgets**. As an example, we developed a toolbar button which displays either an icon, the text or both, depending on the user's preference, with only a few lines of code.
- The **localization** was convenient, although we are convinced that other frameworks will offer similar functionality.
- For coding convenience and efficiency, we appreciated the *ThreadLocal* references to the session variable (*Sessions.getCurrent()*) ZK offers, as repeatedly passing this variable from one function to another is sometimes undesirable.
- ZK can be fine-tuned using configuration files, but they are not required as the developers have chosen reasonable factory defaults.

Scheme 3 (a-d) show some examples of how we could benefit from ZK's flexibility:





Scheme 3: ZK offers the ultimate flexibility we need in science

Only for a few minor detail questions, we needed to find workarounds:

- Passing parameters to included pages, and *reliable, efficient* access thereof (*arg, param, requestScope,...*) was sometimes trial-and-error.
- Databinding with a single binder in the enclosing page and dynamically loaded included ZUL documents, which were also to be bound, was a bit tricky.
- Dynamically generating ZUL elements that integrate into annotation-based databinding was beyond any documented case and required detail understanding of the *AnnotateDatabinder* component. However, due to the published source code of ZK, we could find a good solution for this requirement.

Due to ZK's open source code, we could follow the operations in the debugger and find elegant solutions in many cases. Moreover, the ZK support team and the community provided valuable help and good inspiration for us in many cases.

Summary

Scientific data is very heterogeneous, the future requirements are unpredictable. ZK offers the flexibility we need to manage this kind of data in an efficient way. Only two developers – with only a very limited budget – were able to create an outstanding ELN system, outperforming competitors with legions of developers. This success would not have been possible with the conventional design and development model or any other framework. If I go through the list of Java web application frameworks¹¹, try out demo sites and look at technical documentation from today's perspective, I am still convinced that using ZK was the best option we could choose. A substantial part of our success was only possible as we could take benefit of ZK's great functionality, user-experience and documentation, while relying on ZK's excellent stability and compatibility.

January 2014

Felix Rudolphi, CEO and founder, Sciformation Consulting GmbH

-
- 1 open enventory, AGPL license, <http://open-enventory.eu>
 - 2 http://kluedo.ub.uni-kl.de/files/2262/arbeit_draft.v.18_duplex_b.pdf
 - 3 <http://www.kofo.mpg.de>
 - 4 http://sciformation.com/sciformation_eIn.html?lang=en
 - 5 <http://zkoss.org>
 - 6 <http://www.gwtproject.org>
 - 7 <http://www.gnu.org/licenses/licenses.html>
 - 8 a) <https://jaserverfaces.java.net/>
b) see https://en.wikipedia.org/wiki/JavaServer_Faces#Ajax-enabled_components_and_frameworks for references to selected derivatives and extensions
 - 9 <http://sciformation.com/vectormol.html?lang=en>
 - 10 <http://www.wyzz.info>
 - 11 http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks#Java_2