STORM – Sustainable Online Reporting Model at the University of Oldenburg

Jorge Marx Gómez¹, Sebastian van Vliet¹, Olaf Roeder¹, Swetlana Lipnitskaya¹

Abstract

Sustainability reporting combines economic, environmental and social reporting in one place. Commonly known as “Sustainability Report”, such a report was presented as book or booklet in printed form, which held all information available. Typically, a stakeholder still had to gather information important to him by himself, even from an online available report. The potential offered by “Web 2.0” can vastly improve such information gathering, since it can be presented in a much more individual way.

This is the approach of the Sustain ability Online Reporting Model (STORM), which is a CMS specialized for individual and stakeholder-oriented sustainability reporting. It offers planning, implementation and publication of reports as well as target-group-specific personalization of reports and dialogue with and among stakeholders.

STORM supports the use of GRI indicators and the validation of indicators by a structured scheme. Creating reports, integration of media elements and publication is done within one integrated editor, which greatly simplifies the whole reporting process. The scheme for a report can both be created by companies itself and reused for on-going reporting. Also, an existing scheme can be imported, which could be based on GRI’s reporting guidelines. The personalization of reports are achieved through an information system that allows reordering and picking of certain articles from any report, either on a per user basis or editorially for known stakeholder-groups. These personalized reports can be further distributed, discussed and analysed. STORM archives all reports online and supports distribution via newsletter, e-mail and social networks, making it possible to integrate stakeholders in evaluation, analysing and discussion of reports. Especially machine readable reports support a deep analysis and comparison.

STORM was developed by a student project group at the University of Oldenburg in 2009 – 2010 using only open source software, implemented using the Model-View-Control pattern and additionally divided into several independent modules, which makes it easy to integrate STORM in existing software environments, develop new modules or migrate existing ones to another version or language. The database can be any ODBC compliant database, accessed either locally or remotely; other external systems such as SAP or StudIP (a university’s students organization system) can be accessed, too. Currently STORM needs a J2EE webservice container for operational processing. In this paper, we describe an approach to migrate STORM (as a J2EE Application) to Microsoft’s ASP.NET. This migration will make it possible to run STORM using different technologies, either as independent instances or in a parallel collaborating mode. A long term goal is to fully support (e.g. for import and export) Microsoft’s Office environment and server structures, as those are found in the majority of Small and Medium Enterprises.

The main challenge during that migration is to operate at both platforms simultaneously. To achieve this task, a software-bridge will be implemented, which accumulates gateways between those platforms and manages access. This way functionality is directed at the appropriate target platform, the user interface is kept consistent and the databases are synchronized.

¹ University of Oldenburg, Department of Computing Science, Uhlhornsweg 84, D-26129 Oldenburg, 
jorge.marx.gomez | swetlana.lipnitskaya | sebastian.van.vliet | olaf.roeder@uni-oldenburg.de
1. STORM

STORM is an acronym for SusTainable Online Reporting Model. It combines all necessary steps to create and publish a sustainability report online plus it offers a lot of functionality to further improve the reporting experience, such as personalizing a report for every stakeholder, multiple possibilities of communication (between reporter and stakeholder/stakeholder and stakeholder), evaluation and rating, improvement of future reports, GRI guidelines implementation to support analysis of single reports and comparison of multiple reports.

Basically it offers solutions to two scenarios, first creating reports with multiple authors and information/data providers and second offering all potential stakeholders the possibilities listed above.

STORM was developed by a student project group at the University of Oldenburg in 2009 – 2010 using only open source software, implemented using the Model-View-Control (MVC) pattern and additionally divided into several independent modules, which makes it easy to integrate STORM in existing software environments, develop new modules or to migrate existing ones to another version or language. The database can be any ODBC compliant database, accessed either locally or remotely; other external systems such as SAP or StudIP (a university’s students organization system) can be read, too.

The motivation behind STORM is to move from printed reports where “one size fits all” (Marx Goméz, Isenmann, 2009) to online accessible, individual and dialog-friendly reports. In addition it was planned to use the developed software at the university itself.

2. STORM at the Carl von Ossietzky University Oldenburg

After the first prototype was finished in late 2010, there was an on-going process at the Carl von Ossietzky University of Oldenburg, where several workgroups and related projects made effort to establish a sustainability report. To do so, first STORM had to be further developed which was done by Swetlana Lipnitskaya, Sebastian van Vliet and Olaf Roeder. During that time a lot of research (literature and interviews) was done and used to further improve STORM.

Second, the process of reporting itself had to be established. To do this it was necessary collect information about what information should be reported, where to get that information from and how to collect and work with those information to create a sustainability report.

It was decided, that the majority from the GRI 3.1 indicators plus some indicators from own research should be used. The own indicators are referring to “teaching and research” at universities and were created by another project group in 2010, that cooperated with the STORM project group.

In April 2012 the university’s chairmanship accepted the proposals to publish its own sustainability report. During June STORM was launched in a production environment, so authors and information/data providers could start their work.

As of today, again a group of students is involved in the reporting process (students of the “Sustainability Economics Management”, MA) together with two employees how guarantee completeness and quality, who were also involved in the previous process of establishing the report. The goal is to publish the first report during the first quarter of 2013.

3. Changes and further development of STORM

From 2010 till 2012 there was a lot of process in organization and knowledge gathering concerning STORM and online sustainability reporting, which led to the decision to migrate STORM to another platform. Some of the reasons to migrate STORM are missing support from Grails, originally it was maintained by a company, but later the whole project was passed to the “community”, which means that no official (and thus no guarantee for) support was available any longer. Common problems could be solved
fast with the help of the community, but more special problems needed foreign source code to be investigated and edited, which is a huge workload.

Microsoft's support is not free (as the community's is), but custom solutions for most problems can be given with less workload. Another pro for Microsoft is its distribution among clients, both corporate and private, those people want to keep their Microsoft products.

Another example is the current process of PDF generation. A text in STORM is stored in simple text format, to create a PDF this text is formatted as XML, further processed with XSLT and in a third step to PDF. These three step procedure bring too many problems or side-effects, even though they are automated.

Microsoft offers a platform called Integration-Services, which has a huge library of integration tasks, such as data transformation and integration, event driven tasks, data warehouse functionalities, data mining functionalities, organization of SQL-Servers and more. From interviews with companies and our own experience at the University of Oldenburg we recognized the major problems are as follows, divided in a social and a technical component:

The social problem is the acceptance of a new technology. “Why should I use something new, when my Excel Sheets work perfectly?” To comply with everyone involved in reporting, STORM supports centralized sustainability reporting, to collect all necessary data (e.g. for GRI indicators) and write articles about it, not as a loose collection of text files, but already based on a concrete scheme that fastens publishing in comparison to beforehand mentioned text files, as publishing the final texts is nothing more than a mouse click.

On the publishing site, the biggest benefit is gained, when the next report is to be released, data collection and article lecturing will be a lot easier then, since all requirements are already provided by the system. On the data provider side, it is nothing more than an update, if the data is still provided manually, since it is possible, to get data from external databases, so only the query has to be written once (and then only kept up to date).

On the other side is the technology problem: STORM was originally developed using only open source software, which is a nice approach when creating software for users who want e.g. auditability, low cost and flexibility, but it also has its drawbacks: As for the flexibility, it can be integrated into most software systems, but using a native solution rather than a workaround is the better approach.

Migration of STORM

Since a lot of systems are Microsoft based software systems the STORM developers decided to redesign STORM using Microsoft software. This redesign will be based on the current STORM architecture and functionality, but will use other technologies and put several suggestions into place, to support often wanted features.

The second main goal is to have STORM operating at full functionality the whole time, meaning as the new version is developed, all transferred functionalities should be provided by the new STORM software, where the specific functionality is disabled in the old STORM software.
A software migration can be a very complex task, therefore the migration of STORM will be done using a well-developed method, the Chicken-Little-Migration (Brodie, Stonebraker, 1995). There are several other methods which were considered for this project, but the Chicken-Little (CL) method is the one fitting our needs best.

The CL approach uses small steps to migrate a software. First, the software is analysed (done by most migration methods) to understand the software, clear out bad source code and to make optimizations, if possible and desirable. Second, the software is divided into modules which should be as distinct and low coupling as possible. Next the target environment for each module has to be developed, including databases and interfaces. Finally one module after another is migrated into the new environment. For the migration the architecture of STORM will be kept, but the source code and used technologies will change.

STORM's legacy technologies are Grails, which is a web-development framework using Spring, Hibernate, Groovy and others, MySQL (database), Apache Tomcat Application Server (webserver).

The main difference with STORM's next version is the programming language and environment, which will be Microsoft’s ASP.NET framework (including Common-Language-Runtime, that ensures memory- and thread-management, security, robustness) and Internet-Information-Services (IIS) webserver. Of course STORM will be further developed after the migration is complete.

CL also expects the legacy-software and the migrated software to share resources, therefore gateways are used so that each module – old and new – can operate within the whole software system. A gateway’s task is to connect each module to its correspondent database and to ensure the communication and synchronisation of each module with the whole software environment (Brodie, Stonebraker, 1995). As long as the migration isn’t completed, the gateways are the most important part, since without them the software can’t operate in a mixed state, only the legacy software as is could operate alone (which is not desired in our project).

The benefits of CL over other migration methods are flexibility, security and its gateways. STORM’s migration goes exceptionally well with the CL approach:

- STORM is fully documented, so all modules can be migrated without side-effects or similar problems
- STORM was developed using the Model-View-Control pattern, which makes it easy to migrate each part alone
- In addition STORM has a strong modular architecture, so that these models can be used as CL migration objects without further workload
- Development is on-going with STORM, so it benefits from the gateway techniques

Other migration methods (Cold Turkey (Brodie, Stonebraker, 1995), Butterfly (Bisbal (et al) 1997), Dublo (Hasselbring, Reussner, 2004)) were compared to CL for this project, CL was chosen over the others because of its flexibility, security and its methods and prerequisites (such as parallelism and modularity), that all fit our needs perfectly.
The concrete migration is processed as follows:

Due to CL prerequisites, the legacy software needs to be analysed and modularized, the source code cleaned and the target environment, interfaces etc. defined. The analysis and modularisation can be processed very quickly, because STORM already meets both requirements. The source code clean-up is a task with moderate workload, but can be done quickly with rule-based methods due to Grails’ Convention over Configuration mantra, that also affects source code.

The target environment shares the same architecture with the legacy software, so only some details of ASP.NET implementations need to be configured. Both versions of STORM use the same database technology, but still it is necessary to migrate the data, because Grails and ASP.NET handle object-relational-mapping in different ways. The workload here should be moderate, because only the translation rules have to be created, after that the database migration should experience no problems.

The challenge is, both versions of STORM have to run in a production environment simultaneously, sharing only one view, but both providing complete distinct functionalities.

So a user of STORM only sees one view (usually in his browser window), but this view communicates with two distinct servers to access STORM functions and database via a so-called bridge.

![Figure 1: The GUI-Bridge](image_url)
Figure 1 visualizes how this bridge should work. In short, the view sends its information (e.g. the contents of a form) to the bridge, the bridge “knows” to which version of STORM it has to relay that information. The appropriate version then processes the current task and sends the answer back to the bridge, as if it would be the view. The bridge then interprets this answer and sends it to the real view.

In detail the tasks of the bridge are divided into knowledge of where to send data, this is solved simply with a database, where the target for any action is stored, and the more complex task to translate data. The current version of STORM is implemented in Grails, which uses an HttpServlet object to handle http requests, while the next version of STORM uses Microsoft's HttpHandler objects. The view is implemented with the same technology as the next STORM, so a function call which was already transferred to the new STORM can be easily processed, but a call targeting a function only present in the old STORM has to be translated twice: first when the bridge receives it from the view, it has to translate the call from an HttpHandler object into a HttpServlet object, send it to the Grails' version of STORM, read the HttpServlet object answer and translate it back into an HttpHandler object (and send it back to the view).

In the programmatic context the bridge is a special controller in the MVC-pattern, which filters requests to either send them to the legacy STORM, editing the request object so it can be correctly processed, or relaying the request object to the correct controller within the new implementation.

Once implemented, the bridge's logic doesn't need to be altered, only the database needs to be up-to-date.

4. Conclusion

Since roughly a month STORM is in productive use at the University of Oldenburg, so far only minor technical problems occurred, the acceptance of STORM on the other hand varies, some users provide a lot of positive feedback, but most users didn't give any feedback so far, so it is difficult to evaluate problems. We hope this changes within the next months.

Collecting data is a time-consuming task, since it is the first time being done, where to get the data has been researched already, but most of them still has to be collected manually, since only very few are available in a digital (and even less in a programmatically accessible) form.

Nevertheless we are confident to reach our goal of publishing a sustainability report by the end of the year.

Concerning the migration, for our purpose the bridge is only needed temporarily, but its logic can be reused for other projects, that either need a migration like STORM, or want to combine different technologies or even architectures, without letting the end-user know he is using more than one product at a time.

Bibliography


