Makumba: a Framework for Rapid and Sustainable Web Development by Amateur Programmers

Cristian Bogdan
School of Computer Science and Communication
Royal Institute of Technology, Stockholm, Sweden
cristi@csc.kth.se

Rudolf Mayer
Institute of Software Technology and Interactive Systems
Vienna University of Technology, Austria
mayer@ifs.tuwien.ac.at

ABSTRACT

We present the design of a query-centric web application development framework designed together with and for non-trained programmers. The framework is proven to support sustainable, code-scalable and team-scalable development since 2002. We discuss the positive results of formal qualitative and quantitative evaluation of the framework. We then consider the aspects of the framework design that we believe are at the core of the perceived technology success. Some of these design aspects are, perhaps surprisingly, contrary to current practice in professional web development, which leads us to discuss these contradictions in more detail.

INTRODUCTION AND RELATED WORK

The Intranet of an European-wide organization had just over 600 dynamic pages at its launch in 2002. The Intranet grew steadily in both size and functionality and at the end of 2008 it has almost 2000 dynamic pages. Throughout, except for a few glitches, the system was highly available and fast. Today the Intranet has around 2000 users from among the organization members, and its public part can be accessed by 4000-5000 organization customers at a time.

Such figures would not be surprising for a professional international organization, which hires or outsources professional programmers and system administrators, and can follow a well-defined development process based on established software and web-engineering concepts and techniques. The figures are however unusual for a voluntary student organization, whose IT crew (referred to as the Tech Committee) is made of amateur voluntary student programmers who often do not study Computer Science or related subjects, or are at the early stages of their education. Since 2002, the Tech Committee had over 10 active members at any given time, and the Intranet was extended with more and more subsystems even if about one third of the Tech Committee membership was renewed each year. Reflection and analysis from a technology design perspective on this *sustainability* is the subject matter of this paper. We assert that the design of the technology used to build the Intranet, called Makumba, is a determining factor in this sustainability. We also assert that since a technology worked well in a setting with so little development skill, its relevance for the web development community appears promising.

With the steady rise in importance of web-based applications, the process and tools available for such web-based applications development has been studied intensively. A number of models for the (web-) development process has been proposed, trying to formalize the process with the goal of improving the quality of such web applications. Earlier models include the *Object-Oriented Hypermedia Design Model* (OOHDM) [6], which introduces five different activities in the development process (requirements gathering, conceptual design, navigational design, abstract interface design and implementation), each of which is supported by a set of formalisms and methods, and yields a specific product. The *Web Modeling Language* (WebML) [3] provides a visual notation for web-application design. It defines several different models as different steps in the process, mainly the structural model defining the data, the composition model specifying the content units, and the navigation model establishing the connection between those units. Tool-support for the development process is provided for example with WebRatio [1], which is based on the Eclipse framework. As well as Makumba, WebML allows the user to specify certain queries in the form of the Object Query Language OQL, which significantly simplifies multi-table queries as compared to SQL. Further, WebML allows to define so-called *derived* attributes, which are basically fragments of OQL queries, similar as Makumba supports it via so-called MDD-functions.

While especially WebML provides great support for developing data-intensive web-applications, that might well suit professional settings, the situation is different for voluntary or amateur organizations. In such settings,

CHI 2009, April 4 - 9, 2009, Boston, Massachusetts, USA.

and especially in the distributed development setting detailed below, one major aspect is that in many ways, it is unfeasible to enforce a strict set of processes and work-flows to follow, and specific tools to use, as the amount and timing of work done depends entirely on the motivation and availability of the individual members of the organization. In many cases, applying to strict formalisms might in fact be counter-productive, as it might be perceived as a demotivating restriction on the development work. Also, installation effort required to start contributing should be kept to a minimum, especially considering that a lot of the members of the organization often change their working platform, using their home computer, university equipment, or other systems available while traveling.

We therefore designed a web application development framework which is relaxed in its constraints on formalization, but still enforces a certain development process by it's very own design, and results in web-application which are based on well-known software and web engineering concepts (such as aspect-oriented programming and MVC) by its very nature. Further, we put a specific emphasize on the code-base of the application to be in plain text rather than in visual models or bound to a specific CASE tool or IDE, and thus being allowed to be modified very easily from different operating systems with any kind of text-editing tool available. For the most basic beginners, we also developed a specific browser based tool that provides editing and revision control support, and can be accessed ubiquitously.

Makumba is a framework for developing data-driven web applications, designed for students who do not necessarily know how to program initially. It covers the whole web application spectrum, from creating the relational database tables and indexes, through creating, updating, and removing objects in the database, through calling back in Java application logic, to providing data presentation in HTML. On many of these fronts, Makumba is different from the state of the art, yet its designers have followed closely the evolutions along the years, using technology elements that were considered suitable, such as Java Server Pages and the Tomcat Servlet engine. Still, other technology elements that are usually chosen in the professional Java realm today, such as Struts, Hibernate, Spring, Enterprise Java Beans, Java Data Objects were partially or totally left out of the Makumba design. Web and data-driven application programming patterns like Data Access Object (DAO) and Data Transfer Object (DTO) are not adhered to, while patterns such as Model-View-Controller are partially adhered to and partially circumvented resulting in simpler code, lower code interdependence, better learning curve due to code simplicity, and intuitiveness.

Makumba also has a different programming language focus. Much industrial web development projects are written around procedural or object-oriented languages like Java. While written in Java and its applications employing some Java code, Makumba programming code is mostly constituted by database queries, hence we call it a *query-centric* technology. As we will show in this paper, we regard this ingredient as very important for the success of the Makumba-based Intranet and we are currently building up on this Makumba aspect.

The technological base of Makumba is J2EE[1], which is a well-known framework for enterprise and web application development, building on the Java programming language. While J2EE provides certain assistance for different aspects such as for example data object access and persistence, the complexity of the framework, and its partly lower-level application programmer interface (API) have called for a magnitude of web-application development frameworks building on top of J2EE. An overview of certain shortcomings of J2EE, and how they have been addresses by these frameworks, is given in [4]. Among those, Struts[2] adds support for separating the application in separate concerns using the Model-View-Controller (MVC) architecture, and provides support for building web-based forms.

Hibernate[3] is a object persistence framework, and provides an alternative to the complex Enterprise Java Beans (EJB) framework, which is a part of the J2EE specification. Hibernate is built on idiomatic, transparent persistence of Java objects, and allows to express queries in its own Hibernate Query Language (HQL), an extension of SQL. Makumba can work towards Hibernate-managed databases but it can also work without Hibernate (as it does in its largest application, the Intranet), using a subset of HQL.

Among other popular web application frameworks related to the design of Makumba, but not based on the Java programming language, is for example Ruby on Rails (RoR)[4]. RoR is intended for the agile software development method, focused on rapid application development and short iteration cycles.

In this paper we will reflect on why our design ended up being different from the state of the art, yet still it was successful in this particular setting. We will first introduce the student organization, its group of web application (Intranet) programmers, and the applications they develop and maintain. A presentation of Makumba design follows, with the constraints that shaped it. We then present an evaluation of the technology done with the developers, focusing especially on learning. In the discussion we compare various aspects of Makumba design, learning and use with the corresponding aspects of widely used technologies and patterns for data-driven web development today. We also discuss how some widely used patterns and principles,

[1] http://java.sun.com/javaee/
[2] http://struts.apache.org
[3] http://www.hibernate.org/
[4] http://rubyonrails.org/

like Data Access Object, are not used in Makumba applications, and consider the benefits of this "common practice amendment".

THE SETTING FOR MAKUMBA DESIGN AND USE

The student organization running the Intranet of our interest was grounded in 1988 and is currently present in 80 mainly technical universities across 30 European countries, and has at the moment about 2,500 student members. The Intranet is supporting the activities of the organization, and consists of several integrated subsystems: an application system which registers student applications to the complementary education courses, an internal document archive and member profile management system, a "virtual jobfair" allowing companies to post job adverts, and students to post their profiles. Besides heavily extending and integrating the above sub-systems, several Intranet features were added (using Makumba) since its inception: a Wiki, email archives for the organization's over 500 mailing lists, a training database, a survey-engine, a career-newsletter, and a unique sign-in system that allowed students to share their accounts between various sub-systems, etc. Such new sub-systems, as well as adding more and advanced features to the existing ones, account for the growth of the Intranet code base.

The Tech Committee is the team in charge of developing and maintaining the Intranet, as well as supporting it with activities such as helpdesk. The committee also coordinates activities of interaction design for further new areas of the Intranet. Administration of the Intranet, as well as of communication systems such as e-mail are also Tech Committee responsibilities. In its early (less sustainable) days, the Tech Committee consisted of 1-5 members who were mainly responsible for the individual systems and had difficulties backing each other up when they did not have time for voluntary commitment. Tech Committee membership levels increased after 2002 (constantly around 20), and new members are usually attracted in international meetings with a three-hour Makumba training, including exercises in making dynamic pages that browse Intranet data in various ways, improving existing Intranet pages, followed by assignment of more complex Intranet-related tasks.

It is important to emphasize that the Tech committee members are not formally trained, and are not paid. There are thus specific *motivation* and *learning* aspects in their activity. To be motivated to work without pay in a programming community, a Tech Committee member needs to be able to see very soon that they can contribute, i.e. that they can make a web application element (a JSP page) that displays Intranet data in some new way. Having passed this learning threshold (designed into Makumba to be low) that allows them to get involved, members have still some learning to do, and the essential aspect of such learning is that it is *learning in doing*, without formal training, typically together with peers, or looking at what they have done. During such learning it is important that the members follow a smooth learning path, which, if possible should also be designed into technology [2].

MAKUMBA DESIGN

Design constraints

We will illustrate the constraints that shaped the Makumba design between 1999 and 2001, at a time when few web development frameworks existed. Some of the constraints became apparent when the Tech Committee attempted to port all existing applications to Lotus Notes and Domino, in a bid to reduce the number of technologies that the voluntary team members have to learn in order to maintain and develop the Intranet. This aim was highly motivated given the high change of members actively developing, and the relative short time they stay in the student organization and the Tech Committee.

A framework was needed that can be used by a *highly distributed team*. Most developers work alone, as they are not co-located with other developers. They meet face-to-face a few times a year, but most communication and coordination are carried out online. In this context, the Lotus Notes experience has quickly revealed that graphical modeling and programming techniques are difficult to work with in such a setting: questions that the novice members may have are hard to answer by other developers; communication needs to involve screen-shots or textual descriptions of graphical modeling elements. The use of plain text in other highly distributed programming settings, such as open source projects, is considered as an important factor contributing to the success of these communities [7].

Unlike in a professional development shop, or in an open-source programming setting, developers using Makumba have a wide variety of programming skill levels, some of them not having formal programming training at all. *Accommodating novice programmers* was thus another important design constraint. Makumba needed to assume very little initial knowledge of the developers, sometimes limited to very basic HTML knowledge. A consequence of this has been the important focus on a natural-language inspired programming language, the Structured Query Language (SQL), which can be learned by new members by simply relying on their English. It is possible to build large parts of Makumba systems with only HTML and SQL knowledge. Makumba is thus a *query-centric* framework in its design, i.e. most of its definitions, rules and instructions can be defined in queries. Thus, common to all levels is the usage of a query language, which can either be the Hibernate Query Language (HQL), or Makumba's own Query Language (MQL) which is a HQL subset. Both query languages provide an object-oriented extension to the Structured Query Language, where MQL is explicitly simplifying the syntax for joins between different database tables.

Another consequence of the 'novice programmer' constraint became visible when working with Lotus Notes, which provides developers with long lists of features, in which novice voluntary developers easily got lost. *Providing a small feature set* was thus another principle adopted, which resulted in e.g. a small number of data types offered to Makumba developers. Besides providing a low entry point for novice programmers, it is known that novices often learn by looking at existing code, so it became important to ensure good *code legibility* in a programming unit (JSP page) as well as *easy code navigation* to facilitate finding scripts of interest in a large system as the Intranet.

In the context of the unexperienced developer population, it was not to be expected that many in the developer team would have knowledge of architectural principles for large (web) applications. Therefore there was a need for the framework to ensure *separation of concerns* in the Aspect-oriented Programming (AOP) [5] perspective. This is achieved by using not only different files for data description, data presentation and data modification, but also by using different languages for the respective concerns. This has guided the developer team in maintaining a healthy architecture in the long run, and also served an educational purpose, by acquainting unexperienced developers with the separation of concerns principle. To address this design constraint, Makumba organizes a system in three major parts, the *Makumba Data Definition*, the *JSP level* and the *business logic*.

Makumba Data Definitions

The *Makumba Data Definition* (MDD) describes data structures and relations between them. MDDs are simple lists of data fields, with name and type; the number of data types available is deliberately kept small and to a minimum, to reduce complexity. The most used data types comprise 'primitive types' such as int, real, char, and dates, and 'structural types' such as pointers and sets, which are references to other data types, and enumeration types. An example of an MDD can be seen in Listing 1 for the data type 'Student".

Additionally to defining the mere data structure, Makumba also allows to define validation rules the data has to comply to when created or modified. These rules can be specified in the form of 'query fragments" (expressions in MQL or HQL), and are automatically and transparently composed to complete queries by Makumba. Writing data validation rules in the form of query fragments implies that a great deal of the controller-related tasks can be accomplished without the use of a procedural programming language.

```
name = char[50]
birthdate = date
hobbies = text

education = set
education->name = char[50]
education->university = ptr University
```

Listing 1. Makumba Data Definition "Student"

JSP Level

The *JSP level* is primarily the user-interface level, and is technically a tag library for the Java Server Pages technology, allowing the display and changing of the data stored in a database and described in MDDs. The number of tags needed to perform the most basic operations is again kept very simple. An example of displaying data is given in Listing 2, which shows the code required to display a list of students and, for each student, their completed studies; the data for this list is described in the aforementioned 'Student" MDD (Listing 1).

```
<mak:list from="Student s">
  Name: <mak:value expr="s.name"/> <br/>
  Born on: <mak:value expr="s.birthdate"/><br/>
  Completed studies:
  <mak:list from="s.education e"
            where="e.graduationDate < now()">
    <mak:value expr="e.name"/>
    (<mak:value expr="e.university.name"/>),
  </mak:list>
  <br/>
</mak:list>
```

Listing 2. Example of viewing data with the Makumba JSP tag library

The example uses two tags: mak:list to specify the data definition to retrieve data from, and basic constraints via the 'where' attribute, and mak:value to indicate which fields to display. Notably, composing the query needed to retrieve the data from the database, i.e. the SQL joins and data projections are generated automatically and transparently from notations like 's.education" and 's.surname" respectively. Therefore working with a Makumba JSP would typically not require as much knowledge as using SQL by itself. Note that mak:lists can be nested into each other, and even in that case Makumba will generate exactly one query per mak:list. This optimization leads to a minimum number of queries sent to the database back-end, thereby honoring the single most important performance factor in data-driven web applications today.

Editing objects is done via forms, on a familiar note with HTML, thus helping the programmers in their learning. Listing 3 shows the mak:editForm and mak:input tags. The tags mak:newForm and mak:delete allow for the other typical operations. Rendering of the inputs into text fields, combo-boxes or date-choosers is done automatically, based on the type information provided in the MDD. To edit chains of related objects, forms can be embedded in other forms, as shown in Listing 3.

```
<mak:object from="Student s" where="s=:student">
<mak:editForm object="s" action="somePage" >
  Name: <mak:input field="s.name"/> <br/>
```

```
  Born on: <mak:input field="s.birthdate"/><br/>
  Edit  studies:
   <mak:list from="s.education e">
        <mak:editForm object="e">
          name: <mak:input field="e.name"/><br>
          graduation: <mak:input field="e.graduationDate"/><br>
          university: <mak:input field="e.university"/><br>
        </mak:editForm>
   <mak:list>
  <input type="submit">
<mak:editForm>
</mak:object>
```

Listing 3. Example of changing data with the Makumba JSP tag library

Makumba supports model-driven development methods, as it can automatically create forms and lists by merely specifying the data definition to be used in the form or list, and generating pages by taking information on the data fields from the Makumba Data Definitions; programming-wise, this can be triggered by not specifying the mak:values or mak:inputs in the mak:list and mak:form. Thus, data definition changes are instantly and seamlessly propagated to the JSP level, which makes it ideally suited for Rapid Application Development and fast prototyping. A weaker form of model-driven development support, which allows more influence on the layout of the created page, comes with tools creating the needed JSP pages from the data definitions, applying user-specifiable templates.

Unlike a few other technologies proposed today, Makumba does not employ a navigation model. Instead, the HTML links and form actions are used to determine the next page to execute, thereby building upon its programmers' familiarity with HTML. Extra code to execute for authentication, authorization, validation, and further business rules (described below) is determined automatically according to the data type (MDD) used. This results in a quite scalable design, letting developers create independent pages, while still centralizing the business rules. It is interesting to note that it was at several points considered to 'outsource' some of the Makumba code and features by integrating the Struts technology. It was however deemed that the Struts centralization of action configuration in one single XML file can break the low interdependency between developers that Makumba achieves today, thereby affecting team scalability. It would also have added to the configuration work needed to set up a Makumba application, which is kept to a minimum by design.

Business Logics

The *business logic* (BL) describes, in the Java programming language, "business rules" that restrict the changing of data, and provide authentication and authorization mechanisms. Makumba provides a Java API towards the data base, thus business logics classes can also be utilized to perform processing of complex queries that are harder to generate via the Makumba JSP level. Thus, the simplicity of use of the JSP level does not impose any limit on application functionality. To provide business logic that will be called before e.g. a Student creation, the programmer has to provide a Java method named *on_newStudent()* which will receive as parameters handlers to the page parameters, the session and the database.

Having a complex language like Java to express business logic was however regarded as a shortcoming in relation to the simpler programming languages used at the JSP level. To capitalize on the apparent easiness with which the developers approached query fragments at the JSP level, we decided to attempt to describe as much as possible of business logics using query fragments, directly in MDDs, facilitating code navigation in the process. For example, we recently introduced an *implicit data access layer* that analyzes the queries which require data in the JSPs (through tags such as mak:list) and apply MDD-defined query language functions that are appended (through in-lining) to the generated queries to determine whether the user logged in has access to the respective data. This will replace a lot of Business Logics code that is currently doing the same job at the Java level. Listing 4 shows an example that, once added to the Student MDD in Listing 1. would make the Student object only readable by their owner, or by administrator students. The current Java BL code achieving the same is much more verbose, due to the need of managing a transaction, retrieving an object, preparing a query and its parameters, retrieving its result set, etc.

```
canRead(){actor(Student)=this OR actor(Student).admin=1
```

Listing 4. query fragment authorization rule

This query fragment will then be added transparently to all mak:lists attempting to fetch Student data. In such mak:lists, the programmer can define the default behavior of the containing page, which can be either to filter results which a user is not authorized to view, most suitable for a listing of several data records such as a list of all students, or showing an error message, which is especially useful when showing only one specific account, e.g. a specific member profile.

A large Makumba system: the Intranet

Table 1 shows the evolution of the size of The Intranet since its launch seven years ago, detailed for the different technological levels MDD, JSP and BL. Additionally, the total number of code files is shown, along with the total lines of code (LOC). Figure 1(a) shows the evolution of the code figures graphically. This data indicates quantitatively the project size afforded by Makumba, and the kind of 'healthy' project growth that we argue Makumba can facilitate. The data also illustrates typical proportions of code types in a Makumba project: most code is in JSP scripts made predominantly of HTML and SQL-like code snippets, and there are 6-8 times fewer Java BL (business logic) files, and data

Year	MDD	JSP	BL	# files	LOC	CVS
2002	42	676	80	801	78479	N/A
2003	52	961	116	1132	104805	1143
2004	64	1208	140	1415	127873	702
2005	76	1354	190	1628	151801	1324
2006	99	1719	229	2062	175315	1632
2007	111	2135	287	2559	219456	2391
2008	114	1860	289	2304	196867	1898

Table 1. Size of the Intranet

definition (MDD) files are less than half the number of BLs, and are growing slower in number than JSPs and BLs. There was a rather high amount of MDD files defined in the beginning of the Intranet, representing data types used already in the ancestor systems. Even though more data definitions have been added over time, a certain saturation level with only minor additions was reached in the last years. It is not unexpected that the data definitions of an organization (its "domain model") change seldom after an initial development time.

EVALUATION OF MAKUMBA LEARNING

Throughout the Makumba design, the assumption was that it will be easy for students to read MDDs, and based on that, it will be possible for them to combine HTML and a subset of SQL (itself based on natural language) which they might know prior to start working with Makumba or might learn "on the job". Later on, they would become interested in writing new business rules, and might learn procedural Java for that, maybe using previous knowledge of another procedural programming language. This then would be the "learning path" assumed for a Makumba practitioner.

We designed a questionnaire where 30 out of the 45 past (since 2003) and present members who were approached reflected on their activity in the Tech Committee over their whole 1-4 year-long membership period. Questionnaire results are described in detail in [2], while here we will refer mostly to the questionnaire data on self-assessed learning.

Among the 30 survey respondents, the time of Tech Committee membership ranges from their activity starting around 2001, to fresh members that just joined in the second half of 2008. Also the educational background is diverse – 13 members study computer science, computer engineering or informatics; the other respondents follow various other curricula, ranging from mechanical engineering to physics and biomedical engineering. Also, the membership duration in the committee varies, from 6 members that were active for around four years, to members that were (or currently are) active for one to two years. Most members had no knowledge of Makumba before joining.

To attempt an understanding of the learning aspect of

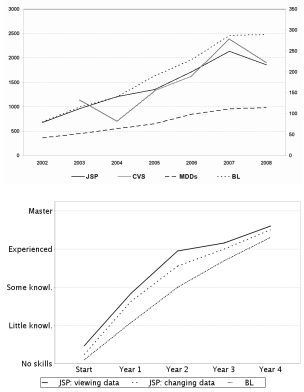

Figure 1. Size of the Intranet (a), learning curve for different aspects of Makumba (b)

Makumba in the context of the Tech Committee, the participants were asked several questions on their self-assessed skill level on database design, SQL, general programming, Java, and three aspects of Makumba: JSP pages to view data, JSP pages to change (create, edit, and delete) data, and business logics (BL). The questions were asked repeatedly for several different points in time, namely when joining the committee, and after the first, second, third and fourth year of membership, and allowed answers from 'no skills' via 'little knowledge', 'some experience' and 'experienced' to 'master'. To balance the self-assessments and to facilitate recall, the participants were further asked to estimate their contribution to the committee in that year, and to list the projects and technologies that they worked with in that year. Also to improve the quality of the self assessment, members were informed that the committee coordinators from their time of activity will review their answers.

Figure 1(b) shows the averaged skill levels of all the members at different points in time, separated for three Makumba usage aspects: JSP for viewing data, JSP for modifying data, and BL. The data suggests that learning to view and display data at the JSP level is faster than handling forms to create, edit, and delete data. Since 64 % of the Intranet files are viewing data, even medium level of Makumba JSP knowledge will give 'access' to approximately two thirds of the Intranet programming units. Learning how to change data has a smaller impact, only 24% of Intranet files are JSPs containing forms. Business Logics are the most difficult concept, especially in the first two years of membership, yet they only cover 12% of the Intranet code. While this Java code is crucial for the Intranet functionality, the novice members have a lot of space to express and make themselves useful elsewhere. The BL code is expected to decrease dramatically when most Intranet business rules will be ported to query fragments of the kind shown in Listing 4. Most members that stay in the committee for a longer period, however, master all levels of Makumba almost equally well. Such members

can (and many do) continue working on the internals of Makumba itself as a next step. While MDD editing skills are less and less actual, we can consider that currently the learning path milestones in the Tech Committee are: JSP viewing data, JSP changing data and BL.

Overall, the per-year self-assessment results suggest that the learning path designed into Makumba has achieved its goal, allowing for members to have an easy start by visualizing data in JSP (which also requires MDD understanding), then starting to change data through JSP forms, and finally learning how to write business rules in Java.

DISCUSSION

As emphasized in our design rationale, and apparently confirmed by our evaluation data, *skill modularity* is a feature that led to the Makumba success in the Intranet project. The first such 'module' has a low learning threshold (HTML and simplified SQL-queries), and the next modules provide developers with new challenge levels (e.g. Business Logic, with the procedural Java skill to be learned).

Especially for the entry-level skill modules (the JSP level of Makumba), *code legibility* is an important Makumba design feature, in such a context of learning in doing, from peers, or from the artifacts they produced. A major part of the Makumba JSP level simplicity and legibility is inherited from the corresponding feature of SQL queries, as legibility and intuitiveness were among its design principles, since it is based on natural language. Members who do not know SQL simply have to rely on their English to find their way around initially. This is further helped by Makumba JSP using a simplified form of SQL queries. Queries and query fragments are currently being introduced in the MDDs, with the aim of reducing the Java BL code, thus further emphasizing the Makumba *query centric* character. Using plain text code rather than visual programming further helps the sharing and understanding of existing examples [7]. While we believe that model driven development is an important and productive movement, our experience runs contrary to many tendencies such as in WebML [3] to use graphical, rather than textual models.

Reflection on Makumba applications architecture leads us to two apparently contradictory considerations. On the one hand, Makumba imposes a widely accepted rule, preventing developers from mixing business rules with data views. Many technologies in use by web developers (like PHP) do not enforce this separation, leading to lack of *code scalability* of application development: application development can be started up fast, like with Makumba, but once applications grow large, problems start to occur due to not enforcing this major principle, affecting in the end sustainability. Of course, trained and experienced programmers can avoid such problems also when using PHP, but this is more unlikely for novice, amateur developers as in our setting.

On the other hand, another principle, the separation of data view from data access is violated by Makumba in the design of its JSP level, since the queries that fetch the data necessary for presentation are part of the presentation units (JSP pages). This issue was emphasized in questionnaire responses by several former developers, currently working with professional IT development. In mixing the data view and the data access, Makumba design ignores the Data Access Object and Data Transfer Object patterns usually employed in the industry, and make Makumba different from other agile development technologies like Ruby on Rails. This however has important code legibility consequences, as programmers only need to work with one file to understand how a unit of the JSP presentation layer works. Also the very code composition phase is eased a lot, allowing the students to contribute fast to the Intranet, which is very important for motivation.

The mix of data access and data presentation also leads to *easy code navigation*. Makumba has fewer types of files than other web application technologies, therefore requiring less navigation between files when performing a task. This helps the novices to make easier progress, and achieve better orientation when contributing to a large project like the Intranet. Furthermore, this source file compactness, especially at the JSP level, leads to little interdependency between the JSP source files. The Intranet could thus grow large in a number of independent directions without much dependency between its JSP units. While the multitude of JSP files still depend on the MDD and BL files, these are almost two orders of magnitude fewer in number, thus reducing the probability for conflict. This *code scalability* helps when several developers work on different parts of the Intranet, as they are not prone to affect each other's work, and thus the likely mistakes that a novice makes will not affect his or her peers. Such *team scalability* at the JSP level is ideal for the sustainability of large web systems development.

As shown by this experience, Makumba allows the building of large Intranets by operating with only the presentation language (HTML for now, but other presentation forms are considered) and a query language. System performance and reliability are high, due to the Makumba query optimization, rivaling professional systems such as Enterprise Java Beans J2EE systems, which are built in much more structured manners, in more complex programming languages like Java. Besides JSP, such systems also employ Java for describing beans, sessions, data access and data transfer objects, and all these are interconnected with a multitude of XML files that tend to interdependencies with many parts of the system, thus reducing code scalability. While such complexity would clearly be beyond the reach of the amateur developers in our case, the Intranet they built can

rival in complexity with many professional J2EE Intranets. This and previous observations question the need for a strict separation of concerns as enforced by J2EE to a large extent and other frameworks like RoR to a lesser extent.

CONCLUSIONS

Based on the Makumba experience we believe that query-centric technologies can gain a firm place among web development frameworks today. Focus on SQL queries, with their later object-oriented counterparts, can lead to good code legibility, little code interdependence, easy code navigation, helping the scalability of the code and the team maintaining it. We also showed that mixing data access and data view can lead to important positive results. We are currently working on data access determined in an implicit, declarative, and also query-centric way.

ACKNOWLEDGMENTS

Thanks to the students, members and associates of the Tech Committee who have worked hard to overcome Makumba imperfections during its and to make the Intranet what it is today, while also taking time to answer our surveys. Thanks are also due to all the Makumba contributors. Yngve Sundblad and John Bowers have supervised this work with good advice during the crucial Makumba design phases.

REFERENCES

1. R. Acerbis, A. Bongio, S. Butti, S. Ceri, F. Ciapessoni, C. Conserva, P. Fraternali, and G. Toffetti. WebRatio, an Innovative Technology for Web Application Development. In *Proceedings of 4th International Conference on Web Engineering (ICWE 2004)*, pages 137–157, Munich, 2004. Springer.

2. C. Bogdan and R. Mayer. Makumba: the role of technology or the sustainability of amateur programming practice and community. In *Proceedings of the 4th Communities and Technologies Conference (C&T 2009)*, Penn State, USA, June 2009. ACM.

3. S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing web sites. In *Proceedings of 9th International World Wide Web Conference*, pages 137–157, Amsterdam, 2000.

4. R. Johnson. J2EE Development Frameworks. *Computer*, 38(1), January 2005.

5. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. marc Loingtier, and J. Irwin. Aspect-oriented programming. In *ProceedingsoftheEuropeanConferenceonObject-OrientedProgramming*, pages 220–242. Springer-Verlag, 1997.

6. D. Schwabe and G. Ross. An object oriented approach to web-based applications design. *Theory and Practice of Object Systems*, 4(4):207–225, 1998.

7. Y. Yamauchi, M. Yokozawa, T. Shinohara, and T. Ishida. Collaboration with lean media: how open-source software succeeds. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 329–338, New York, NY, USA, 2000. ACM.