# Do-It-Yourself Database-Driven Web Applications

Keith Kowalzcykowski
app2you, Inc.
keith@app2you.com

Kian Win Ong
CSE Dept.
UC San Diego
kianwin@ucsd.edu

Kevin Keliang Zhao
CSE Dept.
UC San Diego
kezhao@cs.ucsd.edu

Alin Deutsch
CSE Dept.
UC San Diego
deutsch@cs.ucsd.edu

Yannis Papakonstantinou
CSE Dept.
UC San Diego
yannis@cs.ucsd.edu

Michalis Petropoulos
CSE Dept.
SUNY Buffalo
mpetropo@cse.buffalo.edu

## 1. INTRODUCTION

Originating from research of UCSD's/SDSC's iRods, app2you and FORWARD projects [11][15],[1,2] app2you.com belongs to the emerging space of *Do-It-Yourself (DIY), custom, hosted, database-driven web application platforms* that empower non-programmer business process *owners* to rapidly and cheaply create and evolve applications customized to their organizations' data and process needs. The hoped-for outcome of DIY platforms is paralleled to the emergence of spreadsheets in the 80s and of graphical presentation tools in the 90s [1]. Before their arrival, polished presentations had to be prepared by graphics professionals. PowerPoint enabled us to do them ourselves.

Generally DIY platforms provide an *application design facility* (also called *application specification mechanism*) where the application owner (process owner) specifies the application by manipulating visible aspects of it or by setting configuration options. A simple early example was form builders, where the owner introduces form elements in a form page and the platform, in response, creates a corresponding database schema.

A DIY platform must maximize the following two metrics: First, how wide is its *application scope*, that is, what computation, collaboration on a process, and pages (presentation) can be achieved by applications specified using the platform's design facility? Second, how *easy* is the specification of an application using the platform's design facilities? When the ease increases the technical sophistication required by the owner decreases, and non-programmer process owners are increasingly enabled. The two metrics present an inherent tradeoff. At the one extreme, building applications using Java, Ajax and SQL provides unlimited scope, but does not provide ease of specification. Platforms such as Ruby on Rails [17] and WebML [3] make specification easier and faster, but still not easy enough to enable non-programmer owners. At the other extreme, creating an application by copying an application template, as done for example in Ning [13], is very easy but the scope of the platform is limited to Ning's finite number of templates. DIY platforms are between these two extremes of the scope/ease trade-off (see Section 4 for a discussion of particular ones).

This paper focuses on *database-driven (app2you) web applications*, where users with potentially different roles and rights interact on a web-based process. Depending on the state of the process/application, each user has rights to access certain pages, read certain records in them and execute certain requests. The applications' entire state is captured by the database as opposed to also having out-of-database state that is accessed by the application by interfacing to corresponding external systems.

The *general app2you framework* captures a broad scope of database-driven applications but achieving its full scope requires knowledge of SQL. The *limited app2you framework* is the subset that can by generated by app2you's DIY design facility and is the focus of the paper. The limited framework presents an excellent scope/ease-of-specification tradeoff point: app2you's DIY design facility, which is tuned to the limited framework, enables the easy specification of applications by "business architects" [20], that is, application owners that are not programmers but have the sophistication to reduce their business process into web pages by specifying, in a WYSIWYG fashion and in response to easy-to-understand prompts, properties of the pages such as who can access each page, what is the page's main function, what happens in response to an action.

An advantage of supporting both a general and a limited framework is that even when the non-programmer business process owner cannot create or fully customize the entire web application (typically because parts of the application require the functionality of the general framework) the app2you still enables a much more efficient collaboration between the business process owner and IT specialists: The non-programmer owner creates himself the bulk of the application's pages and workflow, which corresponds to the limited framework, while the IT and specialists provide assistance in elaborate graphics, integration with outside services, code for complex functions , complex SQL and other such aspects that belong to the general framework.


4th *Biennial Conference on Innovative Data Systems Research (CIDR)* January 4-7, 2009, Asilomar, California, USA

[1] Supported by UCSD's von Liebig Center for the commercialization of technology and NSF OCI 0721400.

[2] The license grant at the bottom of the first column does not confer by implication, estoppel or otherwise any license or rights under any patents of authors or The Regents of the University of California.

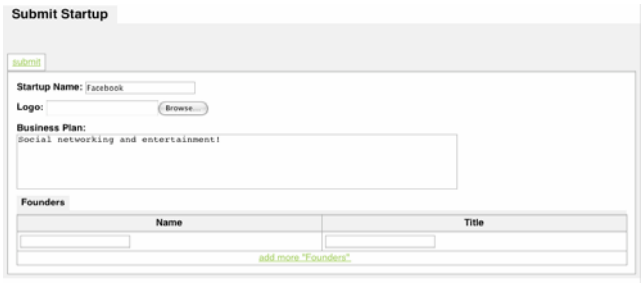


**Figure 1 `Submit Startup` Page**

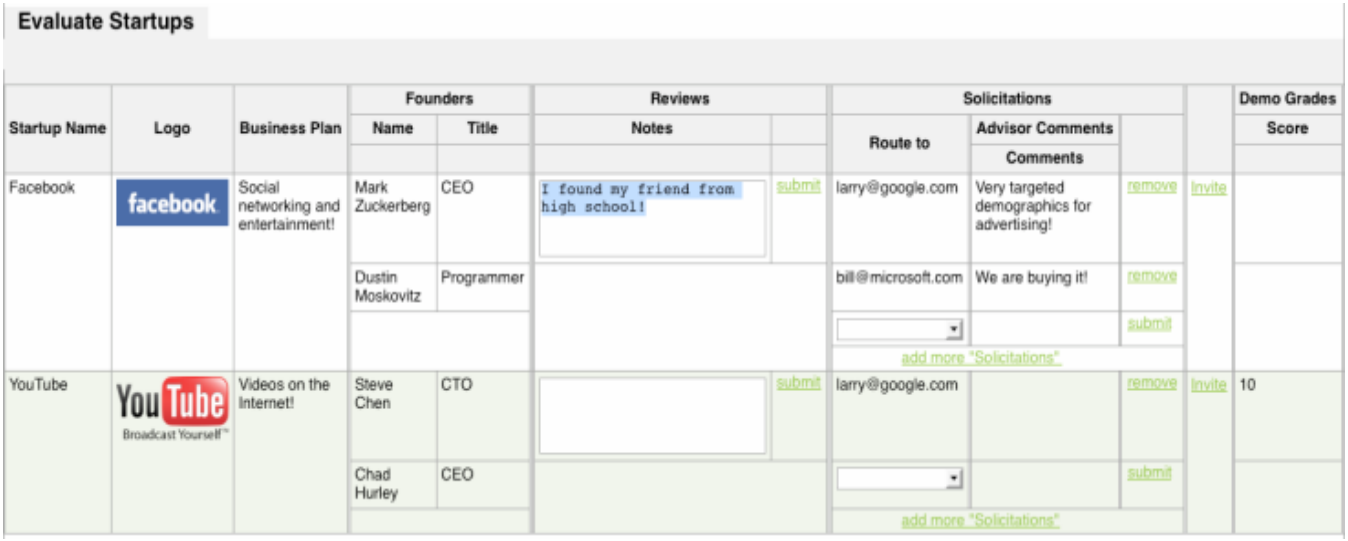


**Figure 2 `Evaluate Startups` Page**

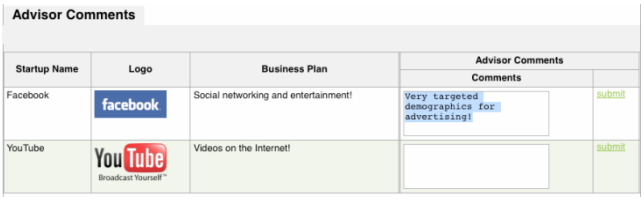


**Figure 3 `Advisor Comments` Page**

Let us first convey informally the scope of limited scope applications through a real-world app2you application. Additional real world examples are found in the Appendix. We use a simplified and modified version of the app2you application for TechCrunch50 (TC50) 2008 [19], a conference where ~1000 startups submitted requests, along with information packages, in order to present themselves and their products. The app2you application was used to collect the submissions, review them and select the top 50 startups. At page `Submit Startup` (Figure 1) any user with a registered account can prepare and submit information regarding her startup, which includes the name, logo, and list of founders.[3] Every user is constrained to at most one startup submission. The submitted startups are displayed on the `Evaluate Startups` page (Figure 2), which is accessible by all reviewers, each of whom can execute three *requests* on each one of them: `submit` a review consisting of `Notes` for each startup; `solicit` comments from one or more advisors, in which case the startup submission will be displayed on the `Advisor Comments` page (Figure 3) to the particular advisors;[4] `invite` the startup submitter to schedule an interview. The invitation results to the submitter receiving an email notifying him to visit the `Schedule Appointment` page, which reports available interview slots, submitted by the reviewers on the `Post Interview Slots` page, and lets the invited startups choose one of them (as summarized in Figure 4). The submitted choices are reported on the `Grade Demo` page, where the reviewers post their grade for the demo given at the agreed interview slot as part of the second review round. Finally, the submitted demo reviews are reported on the `Evaluate Startups` page, where reviewers can now make an informed decision of which 50 startups are the most promising ones.

The general goals of app2you's Do-It-Yourself design facility are typical in easy-to-use systems: (i) WYSIWYG design, where the owner immediately experiences the result of each specification action. (ii) Wizards that suggest to the owner common and semantically meaningful specification options and automate their implementation. (iii) Wizards that explain the specification at a high level where the user does not have to engage in schema design or database queries.

Satisfying such ease-of-use specification goals has required the introduction of multiple novel DIY specification techniques, which are briefly listed in this paper and are further described in [23]. Such techniques have been designed in response to observed stumbling blocks, which we report in this paper, faced by owners in their efforts to build applications.

*Due to the highly interactive, WYSIWYG nature of the DIY design facility we suggest that the reader watches the 10-minute high resolution video at http://www.vimeo.com/2075363, password app2you.*

The first technique is *page-driven design* (Section 3.2), which provides to the owner a WYSIWYG model of the pages. The design facility also contains wizards for specifying properties by answering questions, expressed in easy-to-understand language. app2you in response creates automatically the pages' structure (called *page sketches*, Section 2) and the underlying schemas and queries, therefore relieving the business process owner from designing the database layer, which is one level away from the layers that she understands, namely, the application page layer and the overall workflow. Hiding database schemas, queries, constraints and other low-level details is facilitated by an architecture where high level, easy-to-explain derived properties

[3] Users must first pass from a typical login and signup page before they reach the page of Figure 1.

[4] In the actual application there were solicitations to other reviewers.

of the page sketch hide hard-to-understand complex primitive properties (Section 3.1).

We found out that the inherent difficulty (in comparison to, say, a spreadsheet) in producing a WYSIWYG model for a collaborative application is that the pages typically behave differently depending on which user accesses them and the application state. Resulting enhancements to page-driven design, collectively called *simulation scenarios* ([23]), resolve this problem. They are also demonstrated on the online demo.

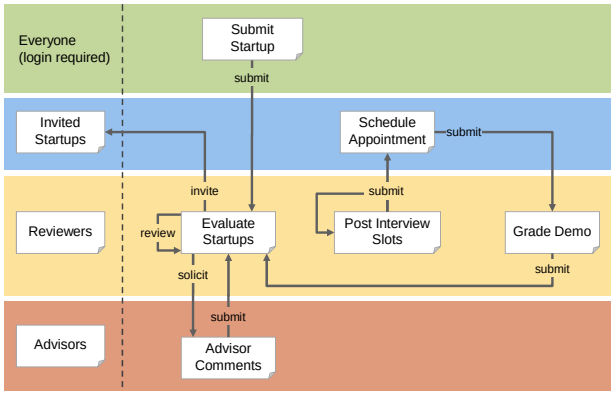


**Figure 4 TechCrunch50 Summary**

Page-driven design by itself still turns out to be insufficient for allowing the owner to reduce a non-trivial multistep workflow she has in mind into a working application. We are in the process of bridging the gap between database-driven applications, designed using the page-driven paradigm, and workflows with a *workflow-driven design extension* ([23]) to page-driven design, which is demonstrated on the online demo. In particular, with the current state of the art, business process workflows are typically specified using user-friendly process-centric models like BPML. These models emphasize and visualize the control flow among sub-tasks of a workflow but unfortunately under-specify the data flow aspects. It is assumed that developers will address the low level aspects, including the specifics of data flow. This situation is obviously unacceptable in a DIY setting for database-driven applications, since there is no developer to take care of the low level details. Using the workflow-driven extensions workflow specifications and visualizations can be produced for database-driven applications that belong to the limited model. Vice versa, we allow the owner to specify applications from a workflow perspective.

Finally, we create a building interface for the *semiautomatic creation of reports*. This interface:

- suggests semantically meaningful joins of various data sets
- requests minimal information from the owner
- discovers the best placement of information on the report

In effect the interface must compensate for the minimality of the owner-provided information with algorithms that make semantically meaningful suggestions and perform complex nested report creation operations. A wide range of static analysis and constraint inference algorithms, drawing from database theory, are utilized for this purpose.

Section 2 presents the part of the general app2you framework that pertains to database-driven applications and the limitations of the limited framework, which are important in enabling the DIY aspect. It argues why both the general and the limited scopes can capture many practical applications. Section 3 briefly describes a few design facility aspects. Section 4 discusses related work.

# 2. FRAMEWORK AND SCOPE

An app2you application is described by its *application sketch*, which is defined by the *general app2you framework*. This paper simplifies the framework presentation by focusing on the part that pertains to database-driven applications, i.e., we ignore here interfacing with external services and systems.

The sketch is modified by the owner when the application is in design mode. The sketch consists of *primitive properties* (collectively called *primitive sketch*) and *derived properties*, where the former are more low level (e.g., queries, constraints) and their settings cannot be derived by the settings of other properties. For ease of specification the non-programmer owner typically does not access the primitive sketch aspects directly, since deconstructing a process into primitive aspects tends to require CS sophistication. Rather the non-programmer owner indirectly accesses them via the *derived properties*, which explain at a high level common questions and options, using wizards and other components of the DIY design facility (Section 3).

The primitive sketch consists of *page sketches*, *user group definitions*, a *database schema* and *general* properties, such as the application name and path.

Each page sketch has a URL, a *page context*, which captures the request parameters (and the types of their values) that are expected upon requesting this page, and a top-level *unit*.

A unit generally has *fields*, a *mode*, *requests* and one *visual template* for each mode. *Atomic fields* generally display data of corresponding parameters of the context.

*Iterator fields* are important for the generation of reports. They have a query, which is typically parameterized by the context *c* and retrieves from the database tuples $t_1,\ldots, t_n$ that have schema *t* and correspond to the records displayed by the iterator. The iterator contains the displayed fields of the retrieved tuples and other components of the application.

For example, the top-level unit of the `Evaluate Startups` page (Figure 2) has an iterator field, whose unit contains the atomic fields `Startup Name`, `Logo`, and `Business Plan`. This iterator runs a query `SELECT * FROM Submit_Startup`, where `Submit_Startup` is the automatically inferred table that collects the non-nested fields of the startup submission form (see Figure 1). It also contains the (nested) iterator field `Founders`, whose unit, in turn, contains the atomic fields `Name` and `Title`. The query of the iterator `Founders` is `SELECT * FROM Founders WHERE Founders.Parent=?` and the parameter (`?`) is instantiated by the `Submit_Startup.ID` of the query result of the containing iterator.

The general framework allows iterator queries to be arbitrary SQL queries over the schema, typically parameterized by values of the context. In this way SQL experts can utilize SQL's full power. The limited framework queries (*lqueries)* are select-(outer)join queries with `EXISTS` predicates, that is, queries of the form `SELECT * FROM` *OuterJoinExpression* `WHERE` *BooleanCondition*, where the condition may be parameterized with values from the context and may also involve `EXISTS(`*SubQuery*`)` predicates where the parameterized *SubQuery* is recursively an lquery. Applications of the limited framework use only lqueries and corresponding constraints, which are generated by the DIY facility. An application however may go outside the limited framework and into the general framework by selectively utilizing "manually" written queries and constraints.

In the future, *calculated fields* will also be associated with queries. In the limited framework such queries will capture the typical

functionality of Excel spreadsheets. In particular, a calculated field may:

1. Compute a new "scalar" value from values of the context. For example, if the context has attributes `First Name` and `Last Name` then the calculated field `Name` may be calculated as `concat(Last Name, ",", First Name)`.
2. Compute an aggregate value by applying an aggregate function over a nested iterator of the page. For example, the non-programmer owner may include in `Evaluate Startups` a calculated field `Number of Founders` that performs the count function over the `Founders` iterator fields.
3. Combinations of the two above.

Lqueries, scalar calculations and aggregates capture the needs of most typical reporting applications and even the needs of relatively unusual request-controlling constraints, such as "each startup may receive at most 5 advisor reviews". Therefore the limitation leads to small scope loss. At the same time, this limitation enables ease of specification benefits: First, the design facility automates the creation of reports for lqueries. Second, filtering and aggregation uses DIY interfaces that have proven themselves in other settings (e.g, spreadsheets). Third, the context created by iterators can be automatically computed, which, in turn becomes important for the automatic inference of the database actions happening when a request is executed [23].

Intuitively, a request combines an HTML input form with information on the effects of submitting the form. A request contains zero or more *input fields*, a button and other properties. The most common effect of executing a request is an update on the database. In the general framework such effect is captured by an SQL statement. In the limited framework the database effect is automatically inferred by the DIY design facility [23]. Note that the inserted/updated records also include *system attributes* such as the auto-generated `ID`, the `submitter` and `creation timestamp` of the record.

Other effects of a request may be (i) sending an email, described by a template (in the style of MS Word mail merge) whose placeholders can refer to both the input fields of the form and the system attributes and (ii) causing a navigation to another page, which can be used to produce confirmation pages and forms submission processes that span multiple pages.

For example, the data submission form of Figure 1 is a request. Its effect is inserting the collected data in tables `Submit_Startup` and `Founders` and sending a confirmation email. The `submit` and `invite` of Figure 2 are the buttons of respective requests, which are called *contextual*, since they happen within the context of `Startup` records.

Note that for DIY simplicity the design facility focuses on pages with a single iterator at the top level unit of the page. Such pages are called report pages. Furthermore, the DIY design is facilitated by *iterator+request field combos* where the iterator part of the combination ranges over data created in response to the request part of the combo. For example, the iterator+request field `Advisor Comments` in Figure 3 combines the `submit` request with an iterator showing the comments collected by the `submit` request.

# 3. DO-IT-YOURSELF DESIGN FACILITY

app2you's design facility allows its non-programmer business process owner to easily and rapidly create a working web application that can be immediately tested and experienced. If the result is not what the owner had in mind a new round of specification-testing can be played out within seconds.

We briefly describe page-driven design (Section 3.2). Further discussion of simulation scenarios, workflow-driven extensions, semiautomatic report creation and details of page-driven design can be found in [23].

At a high level, we use the following principles as a scorecard for the DIY design facility.

- Prefer to provide concrete explanations of sketch properties using WYSIWYG feedback and verbalization of prompts and options that refers to pages, requests and other highly visible properties of the page; rather than being abstract and general.
- Prefer to provide a high-level specification from which primitive properties can be generated, rather than a low-level specification of primitive properties that requires the owner to deconstruct high level concepts into low level concepts.
- Prefer to summarize and enumerate design options to focus on common cases, rather than provide an unstructured, high degree of freedom. "Advanced user", less prominent interfaces should cater to the less common cases.

## 3.1 Derived Properties

Often an important combination of primitive properties must be explained to a non-programmer owner at a high level, which is close to the non-programmer's understanding of the workflow and the function of the pages. Therefore the *derived properties interface* reads the primitive sketch and exports *derived properties* and corresponding common options (called *derived* options) for their settings. When the owner chooses an option the derived properties interface translates it back to the primitive sketch. We describe next a simple example of a derived property, exemplifying the concept. Derived properties become paramount in the following sections.

For example, recall that a user of the `Submit Startup` page may submit only one startup. Once she makes her submission, the form of Figure 1 disappears. At the primitive sketch level, this behavior is achieved by a non-obvious primitive property: The constraint associated with the presence of the form checks that the set of startup submissions of the currently logged-in user is empty. Understanding the behavior of the `Submit` form at this level is fairly complex. Therefore the page wizard offers a derived property asking the much more obvious question of Figure 5.

The combination of a primitive sketch with a derived properties interface produces many benefits on scope and ease of specification:

- It enables the incremental addition of derived properties in the platform, as common cases that lend themselves to higher level explanations emerge, without disrupting existing applications. Indeed, applications created before the introduction of a new derived aspect in the platform can benefit from its introduction: The derived properties interface reads their primitive sketch and exposes a high level derived property.
- It enables a 90/10 rule where the design facility first poses common questions, often relying on derived properties and derived options in order to express them. At the same time, the wide scope enabled by the primitive sketch is available.

## 3.2 Page-Driven Design

The first step towards providing a high-level specification is to allow the process owner to design her application through the WYSIWYG model of pages, as opposed to engaging in low-level web and database programming. Various properties of pages are either specified by direct visualization on the pages, or via answering simple questions about the page. The design facility in

response automatically creates the page's iterator structure, underlying schemas and queries.

Through the high-level specification, page-driven design relieves the owner from specifying data structures in the abstract while en route to construct pages. Moreover, explaining the design options available at the page level promotes easy comprehension, especially if they are explained directly in terms of the application layer that are easily perceived by the owner such as what is the report/form structure of the pages. Lastly, page-driven design facilitates immediate feedback on whether a design satisfies the owner's requirements, since the owner can both inspect and experience the page directly.

### *3.2.1 Page Wizard*

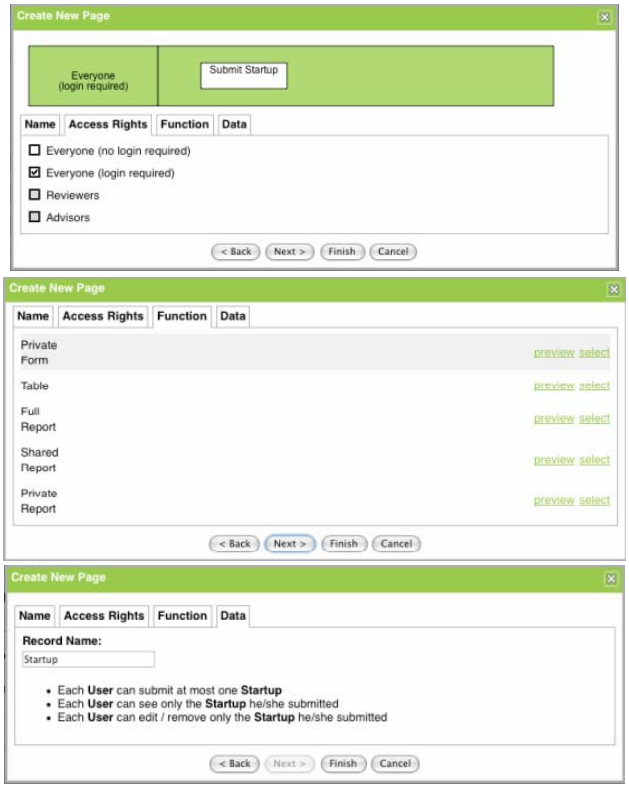


**Figure 5 Page Wizard for `Submit Startup` Page**

The page wizard is the starting point of page-driven design. It prompts with simple questions about page-specific information, such as the page name, URL, and the groups that are authorized to access the page. For example, access to `Submit Startup` is granted to system-defined group `Everyone (no login required)` (Figure 5), whereas access to `Evaluate Startups` is granted to custom group `Reviewers`. Allowing a page to be accessed by a group is also visualized on the workflow diagram by placing the page in the appropriate swim-lane (row).

The page wizard prompts for the main function of the page by enumerating a list of templates, where each template bundles a commonly occurring combination of page properties including presentation format and action rights. Templates are provided to speed up the design of common cases. Such common cases may include forms that allow each user to submit at most one record, and tabular reports where each user sees all records but can only edit/remove the records she submitted, etc. Where the common case is not fully applicable to the scenario at hand, the owner can always customize the page by overriding individual properties independently.

Figure 5 shows the `Private Form` template used in creating the `Submit Startup` page. The template provides the following defaults for the following derived properties:

- The `submit` property of the page's form is set to `on, but max one per user`. (Each applicant can only submit one startup.)
- The `display` property of the page's iterator is set to `on if user has submitted the record, off otherwise`. (Each applicant can only see the startup info she has submitted.)
- The `edit` and `remove` properties of the page's iterator are also set to `on if user has submitted the record, off otherwise`. (Each applicant can only edit or remove the startup info she has submitted)

Whenever the `submit` aspect is `on`, the page wizard also prompts the owner to optionally assign a name to the records collected. The record name helps the system phrase questions and options more specifically. Figure 5 shows the wizard for `Submit Startup.` It starts with a system-proposed default of `Record submitted at Submit Startup`, which is later set by the owner to `Startup`.

### *3.2.2 WYSIWYG Design*

After the basic properties of the page have been specified through the page wizard, the owner can customize the form of the page in a WYSIWYG fashion. To create new input fields, the owner drags-and-drop input components such as text boxes, image upload prompts, dropdown boxes, check boxes etc. into the request form of the page (Figure 6).

For each input component dragged into the form, a corresponding field is added to the request, and a corresponding attribute is added to the schema of the database table where the records corresponding to the request are inserted. The input component determines the data type of the field. For example, the `Logo` field is created through an image upload component, therefore storage is allocated for binary data, data can be submitted through an HTML file input form element, and submitted data are displayed as images.

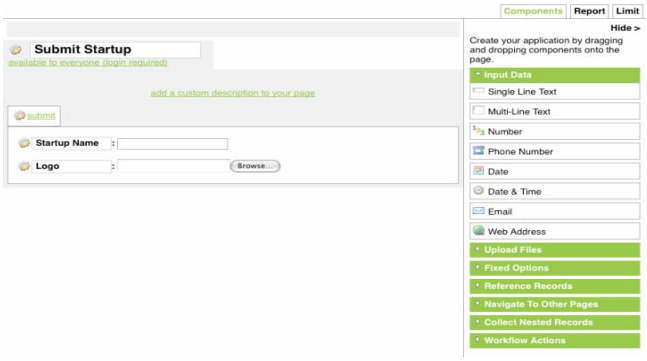


**Figure 6 WYSIWYG Page Design**

The owner may also introduce repeating nested data by creating nested tables, such as the `Founders` on the `Submit Startup` page. Besides plain forms or plain reports the owner can also design pages that have forms in the context of reports [23].

## 4. RELATED WORK

The technological background for Do-It-Yourself database-driven applications is fertile for two reasons. First, they leverage the emergence of hosted applications (software as a service) and Web 2.0 Ajax-based interfaces that allow application page design from

the comfort of one's browser, while providing the richness of desktop interfaces. The two aspects combine to remove the hassles of (i) downloading/installing software in order to create an application and (ii) deploying/exporting an application on the web. But the Do-It-Yourself ability presents a larger, qualitatively-different challenge: How to disrupt conventional database-driven web application programming by providing brand new models of specifying database-driven web applications so that non-programmer business owners can build their own applications.

Multiple systems support the fast creation of custom web applications by removing the need to program in a complex Turing-complete programming language, such as Java. WebML [3] is a prime example of *schema-driven application creation* (also see DeClarit [6], Oracle Express [14]). The creator starts by designing the Entity-Relationship data model for her application. Then it is easy to specify pages by putting together units that accomplish typical functionalities of Web applications. For example, a unit may report the data of an entity and utilize the relationships of the data model to navigate to related entities. It is reported [22] that the development and maintenance of WebML applications led to 30% increased productivity with 46 distinct applications maintained by 5 part-time, junior developers.

The emerging Do-It-Yourself custom application platforms primarily target non-programmer process owners. A common theme is that the owner does not need to create a database schema in the abstract. Rather she builds forms, which automatically lead to corresponding tables that are typically reported on the same page. Such systems tend to be *online databases* [4][5][7][9] for easy information sharing and collaboration, often delivering great advantages over online spreadsheets, which are their main competitor for structured information sharing[5]. However, the resulting applications have a very limited scope (and business logic): Users simply post and read structured data in the shared space.

A next generation of Do-It-Yourself systems promises to go beyond information sharing and to enable users to capture their business processes by web applications. At a high level, these platforms are either "MS Access online" [4][2] or customizable vertical templates [18].

The "MS Access online" enablers allow users to create multiple Do-It-Yourself online tables (having forms and reports to give access to them). In the same spirit with MS Access, the reports have to be fueled by queries where the user has explicitly specified joins and selections. Finally, business logic and flow of data from table to table is offered in the form of scripting programming languages [12] or graphical languages [4] that allow the user to describe series of insertions, deletions and updates and the conditions under which they should happen. The adherence to tables with separate forms and reports creates problems at both the scope axis and the easy specification: The web applications we are dealing with day-to-day are not mere collections of tables with a report and a form for each table. A typical case is that the input forms of a page typically operate within the context of reported dynamic data and even within the context that prior pages create, i.e., there is no artificial divide of "input only" and "report only", as is clearly evidenced by pages such as `Evaluate Startups` and `Advisor Comments`. In addition to app2you, AppForge [2] also solves this problem.

[5] Yahoo Pipes [21] and IBM's QEDWiki [8] represent high end versions of the information sharing space, where data from multiple sources and RSS feeds can be automatically integrated and presented online.

Another scope problem of "MS Access Online" is the inability to capture that access rights to a page may depend on the business logic itself. For example, in the TC50 application the group "Invited Applicants" is derived automatically and controls access to "Schedule an Appointment".

The "MS Access online" class is problematic in creating workflow application since the business process owner needs to reduce the collaborative process she has in her mind into normalized tables and into sophisticated queries and updates. For example, we discussed how hard it is to explain using a query that the `Advisor Comments` should only show startup submissions that have been passed to the currently logged-in user. This raises the bar of sophistication needed by the builders towards the level of sophistication that programmers have, therefore seriously limiting who can create and evolve applications. The anecdotal evidence behind this thesis is plenty: Instructors of undergraduate database classes know the difficulty that, even computer science students, have in designing appropriate schemas and writing non-trivial queries. Furthermore, despite the best efforts of tools, such as the tools of the Microsoft Office Access and Microsoft InfoPath, to make database schema design and query writing approachable by the masses, the general public has found it hard to engage in those activities. The above evidence is not surprising since database schemas and queries are abstract structures that have no immediately visible connection to the web application and workflow aspects that the non-sophisticated designer can immediately associate with, which are the Web pages with which the users of the application will be interacting.

Applications with fixed workflow and database table structure and customizable input form structure (i.e., one can change the attributes of tables as long as the tables and their interactions remain fixed) have been a great success [18]. We believe that customization does not need to stop at that point since, by doing so, the scope of available applications is limited by the available initial templates.

## 5. REFERENCES


[1] Jeanette Borzo: Do-It-Yourself Software, Wall Street Journal, 9/24/2007. http://online.wsj.com/article/SB119023041951932741.html#articleTabs%3Darticle.

[2] Chavdar Botev, Nitin Gupta, Jayavel Shanmugasundaram, Fan Yang: A WYSIWYG Development Platform for Data Driven Web Applications. VLDB 2008.

[3] Stefano Ceri, Piero Fraternali, Aldo Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33(1-6): 137-157 (2000).

[4] Coghead. http://www.coghead.com

[5] DabbleDB. http://www.dabbledb.com

[6] DeKlarit. http://www.deklarit.com

[7] eUnifyDB. http://www.eunifydb.net

[8] IBM QEDWiki. http://services.alphaworks.ibm.com/qedwiki/

[9] Intuit Quickbase. http://www.quickbase.com

[10] JavaServer Pages Technology http://java.sun.com/products/jsp/index.jsp

[11] K.W. Ong, Y. Papakonstantinou, K.K Zhao: Do-It-Yourself Forms-Driven & Workflow Database-Driven Applications. Provisional patent submitted by University of California at San Diego, December 2008.

[12] LongJump. http://longjump.com

[13] Ning. http://www.ning.com

[14] Oracle Application Express. http://www.oracle.com/technology/products/database/application_express/index.html

[15] Y. Papakonstantinou, I. Katsis, K. Ong: Creating Hosted Web Application and database. Utility patent submitted by University of California at San Diego, April 2007

[16] Lucian Popa, Alin Deutsch, Arnaud Sahuguet, Val Tannen: A Chase Too Far? SIGMOD Conference 2000: 273-284.

[17] Ruby on Rails. http://www.rubyonrails.org/

[18] Salesforce.com. http://www.salesforce.com

[19] TechCrunch50 (TC50). http://techcrunch50.com

[20] Colin Teubner and Ken Vollmer: BPMS Revenue To Reach $6.3 Billion By 2011. Forrester Research, 2007. http://db.ucsd.edu/app2you/2009-www/2007-forrester-bpms.pdf

[21] Yahoo! Pipes. http://pipes.yahoo.com/pipes/

[22] Piero Fraternali, Stefano Ceri, Massimo Tisi: Developing eBusiness solutions with a Model Driven Approach. IMP 2006.

[23] K. Kowalczykowski, K. W. Ong, K. K. Zhao, A. Deutsch, Y. Papakonstantinou, M. Petropoulos : Do-It-Yourself Database-Driven Web Applications (Extended Version) `http://db.ucsd.edu/pubsFileFolder/319.pdf`

## Appendix

More than twenty forms-driven applications have been built and used in 2008 on app2you.com. For example, a recruiter has collected job openings from its customers. A wide group of users, defined and controlled by the recruiter, sees selected fields of the job openings' records and is invited to recommend individuals, who are notified about the positions, provide their level of interest and proceed to exchange information with the customer and the recruiter if interested.

In another example, the United Cerebral Palsy non-profit organization maintains an online loan library of toys, keeping track of who currently holds a toy and who has requested it.

In multiple variations of classroom management applications students submit their projects, often after a phase where they have teamed up in project teams. The TAs and instructor provide feedback and grade. Variations include setting up appointments for project presentations and rehearsals, voting for the best project etc.