Social Information Systems
The End of Shadow Applications?

Marc Quast[1] and Mark J. Handel[2]
[1]*University of Grenoble, Campus C207, 220 Rue de la Chimie, 38400 Saint-Martin d'Hères, France*
[2]*The Boeing Company, MC 7L-70, PO Box 3707, Seattle, WA 98124, U.S.A.*

Keywords: Information Systems, Business Applications, Enterprise Architecture, Social Software Engineering.

Abstract: In large corporations, line-of-business organizations frequently introduce unofficial "shadow" applications to work around the limitations of the established information system. This paper presents a software architecture designed to alleviate this phenomenon, and reuses examples from a recent industry experience report to demonstrate how shadow application proliferation could be avoided without sacrificing flexibility and reactivity. We present the initial results of our prototype, and discuss the possibility of a social information system designed to both reduce the present chaos and enable the cooperative design and evolution of business applications.

1 INTRODUCTION

Delivering the right information at the right time to the right persons is one of the most important requirements of today's business world (Spahn and Wulf, 2009). Nevertheless, corporate information systems are a widespread source of frustration (Newell et al., 2007). Business units do not accept the poor service provided by their IT departments and build up independent IT resources to suit their specific or urgent requirements (Zarnekow et al., 2006).

As a result, information systems of large corporations are a web of numerous applications. At the center we find a fairly small set of stable and robust enterprise applications. These are surrounded by a larger set of semi-official applications and a very large number of unofficial applications. We adopt the term of *shadow application* proposed by Handel and Poltrock (2010) for the last two categories, i.e. applications introduced by business units to satisfy requirements not met by official applications.

Though the benefit of "getting the job done" is sufficient to justify, and indeed pay for, their existence, shadow applications raise serious problems: duplicated and inconsistent data is commonplace, and having critical information and functionality scattered, unreachable and managed outside of standard IT processes is obviously not what comes to mind when envisioning a well-structured and robust information system.

Building upon our industry experience[1], this paper proposes a potential solution. After a short definition of shadow applications, their main characteristics and the causes of their emergence, we propose an alternative architecture for business applications which could prevent the systematic recourse to shadow applications in their vicinity, using two use cases from (Handel and Poltrock, 2010) to illustrate its effects. We present our prototype implementation and our first results, and discuss the possibility of a social information system designed to both reduce the present chaos and enable the cooperative design and evolution of business applications.

2 UNDERSTANDING SHADOW APPLICATIONS

Shadow applications are characterized by their purpose. If application A exists to work around the limitations of application B, or if A's features belong in B according to its users, A can be considered a

[1]The authors have a cumulated experience of over thirty years in the development and operation of business applications in industrial environments.

shadow application. This partial definition illustrates the subjective nature of the phenomenon.

Shadow applications are also characterized by their ownership. If it is owned by the IT department, it is an official application; otherwise it is a shadow application. The important distinction is not so much "IT or not IT" but "ownership by the actor effectively using the application". This allows the owner to quickly adapt the tool without consulting other parties or relying on the IT organization's priorities. It also provides him with full control over the visibility of the data and access to features.

Individual spreadsheets meet this definition. These are often used for simple data storage and manipulation, as a substitute for more robust business applications. This is a very common and possibly dominant use case since their introduction (Nardi and Miller, 1990), and Handel and Poltrock (2010) qualify such spreadsheets as shadow applications.

"Official" and "shadow" are relative concepts, and apply recursively at various levels of an organization. In other terms, multiple layers of shadow applications exist, the final one being personal applications.

Shadow applications are typically loosely integrated with some official and other shadow applications. However, manual synchronization is not uncommon (Hordijk and Wieringa, 2010).

We define a shadow application as an application which:

- works around another application's limitations and
- is both functionally and technically owned by the organization using it.

Shadow applications are usually considered a "necessary evil" (Hordijk and Wieringa, 2010). Organizations cannot work without them, but would prefer to avoid the data duplication they imply as well as the burden they represent in development and maintenance costs.

The benefits of shadow applications must outweigh the drawbacks; otherwise line-of-business organizations would not develop, deploy, and maintain them. We will refer to the main benefits of shadow applications as perceived by their owners as the "AVI capabilities."

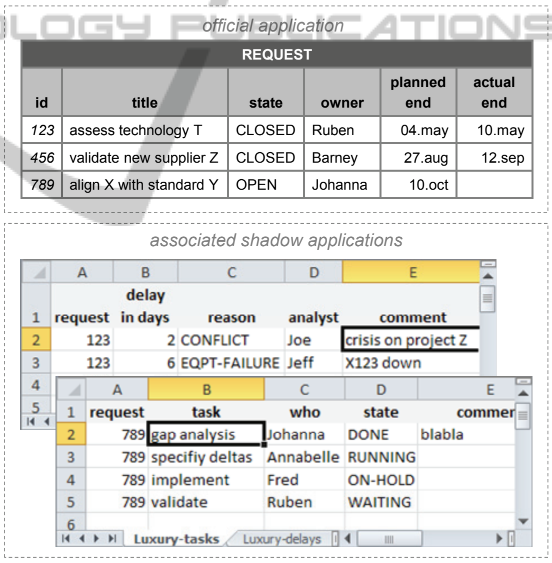
- The owner has full Autonomy to implement new features.
- The owner decides about Visibility of the application to the larger organization.
- The owner can Integrate (manually or automatically) with other applications.

2.1 Examples of Shadow Applications

A recent experience report contributes observations about shadow applications in a 10 year engineering project (Handel and Poltrock, 2010). In this paper, we will use fictional examples derived from the information disclosed in this report.

- *"Luxury can report delays on process instances, but not the reasons for these delays which are managed by a shadow application."*
- *"Sometimes the tasks tracked by Luxury were informally decomposed into subtasks; (...) Luxury had no provisions for this kind of task decomposition."*

We make the assumption that Luxury tracks requests, a common use case in engineering environments. Figure 1 below shows a fictional central database of an official application and two of its shadow applications, managing delay analyses and subtasks respectively.



REQUEST

id	title	state	owner	planned end	actual end
123	assess technology T	CLOSED	Ruben	04.may	10.may
456	validate new supplier Z	CLOSED	Barney	27.aug	12.sep
789	align X with standard Y	OPEN	Johanna	10.oct	

	A	B	C	D	E
1	request	delay in days	reason	analyst	comment
2	123	2	CONFLICT	Joe	crisis on project Z
3	123	6	EQPT-FAILURE	Jeff	X123 down

	A	B	C	D	E
1	request	task	who	state	commer
2	789	gap analysis	Johanna	DONE	blabla
3	789	specifiy deltas	Annabelle	RUNNING	
4	789	implement	Fred	ON-HOLD	
5	789	validate	Ruben	WAITING	
6					

Figure 1: Example of fictional official database and associated shadow application data.

While spreadsheets are arguably the most common form, shadow application architectures are limited only by the owner's resources, including full-blown business applications and, more fashionably, third-party applications in the "stealth cloud", i.e. cloud services being consumed by business users without the knowledge, permission or support of the IT department (Gotts, 2010.).

2.2 Causes of Shadow Application Emergence

Shadow applications emerge to work around the shortcomings of official applications (Zarnekow et al., 2006). Thus we need to understand the causes for these problems.

Large organizations are not consistent and orderly systems. Referring both to groups and individuals, Kling (1991) describes working relationships as *"multivalent with and mix elements of cooperation, conflict, conviviality, competition, collaboration, control, coercion, coordination and combat (the c-words)"*. Requirements from different stakeholders are thus often divergent or conflicting, which explains why the difficulty of requirements engineering increases exponentially with the number and diversity of participants. Ackerman (2000) indicates that when there are hidden or conflicting goals, people will resist articulating these. Under such circumstances, it is a challenge to converge on a consistent set of requirements and deliver a working application at all. But widespread dissatisfaction with the result is almost guaranteed by construction.

As an aggravating factor, corporations are not static. They must adapt to changes in their environment like new markets, technologies or regulations. Though the aforementioned c-words impact is often obvious at the time of application introduction, the continuous evolution of business requirements turns this into a subtle though continuous problem. Any change in any stakeholder's universe can invalidate the initial compromises and demand new rounds of discussion, yielding further dissatisfaction.

Besides inter-organization conflicts, some c-words foster shadow application emergence by themselves. A successful shadow application and the knowledge it captures is usually highly visible within an organization, and its ownership provides recognition (competition) and power (control, coercion).

There are other contributing factors. The widespread practice of reducing IT costs lowers both reactivity and quality of IT support, inciting business units to help themselves (Hoyer and Stanoevska-Slabena, 2008). Technical obsolescence, a consequence of either respectable age or unfortunate choice of foundation technologies, can make it difficult to find the right skillset to implement changes. This paper focuses on the following factors leading to shadow application emergence.

- Business unit considers it impossible to converge on a single set of requirements fulfilling all stakeholders' requirements.
- Business unit does not want to rely on slow or expensive third parties.
- Business unit considers it in its best interest to produce a new system they own.

2.3 Preventing Shadow Application Proliferation

Our opinion is that with present software architectures, no matter how carefully official applications are crafted, over time they will spawn shadow applications whenever resourceful communities have urgent unsatisfied needs.

Our hypothesis is that if an application provides the AVI capabilities, the need for shadow applications is greatly reduced. Today's software architectures cannot provide these capabilities because the components of a business application (such as data elements, workflows, or forms) are *shared* among organizations. This sharing is both the main reason why business applications exist and the main reason for the emergence of their shadow counterparts. We therefore propose an application architecture with a fundamentally new and different sharing principle.

3 REQUIREMENTS FOR AN ALTERNATIVE APPLICATION ARCHITECTURE

In this section we attempt to express the AVI capabilities as a set of requirements for an application architecture, with the following definitions.

- *Actor* designates an individual or a group of individuals, for example the entire company, an organization, department, project team, or community.
- *Elements* are runtime application components, like business entities (in our previous example a "*Request*"), workflows, forms, reports, or even configuration entries. With this definition an *Application* is a collection of related Elements.

3.1 Functional Requirements

Our first two requirements cover the most central operations in shadow application development.

R1: *Actors can extend existing Elements.*

R2: *Actors can add new Elements.*

Example 1 below reuses an observation from (Handel and Poltrock, 2010) to illustrate how an application satisfying R1 and R2 could defuse the need for shadow applications.

Example 1 – *Luxury*[2]

The official application manages *Request* entities, with among others attributes *title*, *state* and *delay*.

The "Quality" department needs to record the reasons for delays when they occur. Using R2, they introduce a new Element *DelayAnalysis* with attributes like *reasonForDelay* and *analyst* and associations with existing Element *Request*. Behind the scene, this leverages R1 to extend the Request Element with the reverse association *delayAnalyses*. This blends the new Element and extensions with the original *Luxury* entities thus enabling intuitive bi-directional navigation.

Other operations are adding missing attributes to an existing business element or adding more detailed states in an existing workflow. Example 1 highlights a new problem: the extensions are of interest only to a subset of the application's users, and may be confidential. To avoid cross-Actor pollution and conflicts, both R1 and R2 imply that Actors are *isolated* from each other by default, which yields the requirement R3.

R3: *Actors have private spaces.*

Elements are hosted in such private spaces and are by default not visible outside of them. We call these spaces *Perspectives*. In a typical enterprise setting, today's official applications would be Perspectives providing 'scaffolding' Elements, i.e. skeletons of business entities and associated high-level rules and functionality. Organizations at various levels would have their own Perspectives, hosting the extensions and additional Elements reflecting their concerns and level of detail. Individuals could likewise replace their spreadsheets with private extensions and Elements hosted in a private Perspective. However, completely isolated Perspectives would defeat the purpose of enterprise applications, which yields R4.

R4: *Actors can share the Elements they own.*

[2]In the report, *Luxury* refers to both a business process and the supporting official application(s). We only refer to the latter here.

Perspectives can make selected Elements visible, either to everybody ("public") or to a restricted set of Perspectives. We call this operation *export*. Obviously the previously mentioned official Perspectives would export their Elements to all users. And business-unit-level Perspectives would export their Elements to the relevant Actors. Even individuals can share their Elements with others.

It is interesting to note here that this empowers the entire employee base to contribute to the overall information system, which we think provides significant benefits we will discuss later in this paper. The downside is that this could lead to cacophony through an overwhelming amount of available Elements, dictating R5.

R5: *Actors can select relevant Elements.*

Thus, a symmetrical *import* operation is necessary. An Actor must be able to select among all Elements available to him only the ones he considers relevant. Instead of building his environment from scratch an Actor would *inherit* the Elements from the groups he belongs to, but must be able to *unimport* these if not relevant for him. Example 2 below, again from Handel and Poltrock (2010), illustrates how R1-R5 could have avoided another real-life shadow application.

Example 2 – *Fallen*

"Official application *Fallen* had produced a shadow application which added translations into Japanese next to English data fields."

Extending existing entities with additional attributes is a typical use case of R1. Such extensions would be owned by the Japanese branch of the company, and hosted on their servers in a Perspective (R3) we can call *http://fallen.acme.co.jp/Translations*.

Employees of the Japanese branch would inherit these extensions, and some groups or Japanese employees could even choose to unimport the initial English attributes (R5). The extensions could be exported to other Japanese-spoken employees in other regions (R4).

Our previous use of the term Application encompassed a broad spectrum, from full-blown enterprise systems to private spreadsheets. Likewise, for Perspectives we envision a broad range from big Perspectives hosting self-sufficient third-party applications to tiny individual Perspectives with just a few extensions replacing spreadsheets. Some Perspectives may just factorize the optimal list of import and unimport declarations for a given organization or community.

3.2 Usability Requirements

A significant percentage of today's shadow applications are created by people without software engineering skills using office software like spreadsheets (Nardi and Miller, 1990). This observation makes usability a key requirement.

R6: *No programming is required for R1-R5.*

The last item may sound like reviving the dream of software without programmers. However, the data-centric nature of business applications makes it much less difficult for end-users to participate than more feature-centric software; significant contributions of entities, attributes, simple formulas and associations can be made through a forms-based interface, especially in the presence of example instances (Markl, Altinel, Simmen and Singh, 2008).

Contributions are not limited to what can be done by end-users through forms. A language-based representation of perspectives and elements is still necessary for professional software developers. Even for business units, contractors and interns have always been a means to get access to development skills beyond their internal competence to implement complex shadow applications. In a perspective-centric architecture, such expert contributions would still be possible, with the benefit of being better integrated with the rest of the information system.

4 CONCEPTUALIZATION

Figure 2 below shows the high-level meta-model of our proposal. It is centered on a classical enterprise directory component with *Users* and *Groups*. *Perspectives* are hosted by *Repositories*. Perspectives can define *Fragments*, which can be either self-sufficient (R2) or extensions of a Fragment from another Perspective (R1). Repositories can live on different servers.

At runtime, a User opens a *Session*, which determines a set of Perspectives – owned by the User or inherited from the Groups he is member of. This in turn determines a set of Fragments, which can be woven into *Elements*. The Session becomes the Application, tailored to the connected user's profile. We call this a *virtual private application*, private because it reflects the user's unique combination of elements, virtual because it does not exist outside of the session.

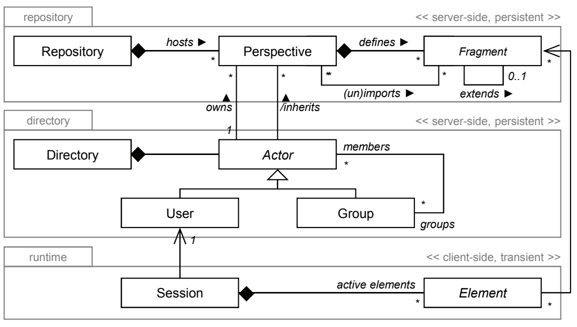


Figure 2: Meta-model of Perspective-centric architecture.

In a perspective-centric architecture, applications are thus dynamically composed at runtime. Today, commercial-off-the-shelf (COTS) applications need to suit the requirements of a variety of customers and provide some degree of flexibility through configuration and customization mechanisms (Brehm, Heinzl, and Markus, 2001). We consider our proposal a *generalization* of these mechanisms found in application platforms like issue-tracking, PLM and ERP systems.

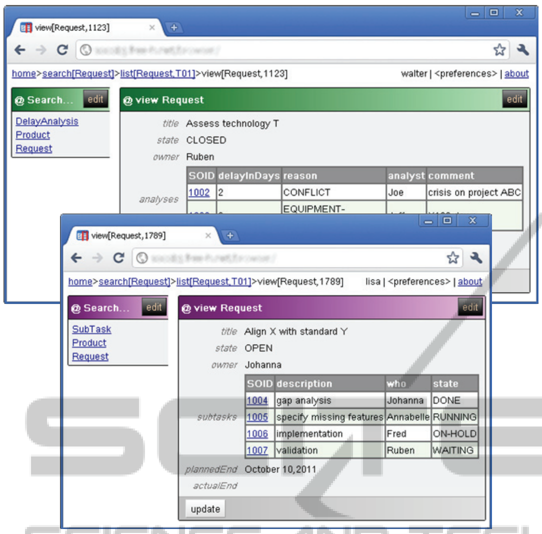
5 PROTOTYPE

We have designed and implemented a first prototype of a perspective-centric system. Considering the complexity of the general case of extensible Elements, our prototype mainly focuses on data, i.e. business entities.

5.1 End-user Experience

Our main objective was to verify that the dynamic, perspective-centric nature of the system could be made transparent to end-users during normal use. The screenshots below show two different users connected to a Luxury-like application, both displaying a request object. The first user belongs to the quality group and thus sees DelayAnalysis objects, the second user is from the planning group and sees SubTask objects.

It is important to stress the *additive* nature of the system, as opposed to subtractive, i.e. filtering. In a filtering approach, somewhere an Element would exist with all attributes, which get filtered out depending on the users' profile. In our approach, Fragments exist in various places and get pulled together by the Session. The main visible difference with a regular system is the presence of edit buttons, which allow inspection and tailoring of the connected user's model as illustrated in the next screenshot, which shows (1) the possibility to import another Entity "Customer", and (2) that Element "Request" is a composition of Fragments from three different perspectives.


view[Request,1123]

home>search[Request]>list[Request,T01]>view[Request,1123] walter | <preferences> | about

@ Search... edit

DelayAnalysis
Product
Request

@ view Request edit

title Assess technology T
state CLOSED
owner Ruben

analyses

SOID	delayInDays	reason	analyst	comment
1002	2	CONFLICT	Joe	crisis on project ABC
		EQUIPMENT-		

view[Request,1789]

home>search[Request]>list[Request,T01]>view[Request,1789] lisa | <preferences> | about

@ Search... edit

SubTask
Product
Request

@ view Request edit

title Align X with standard Y
state OPEN
owner Johanna

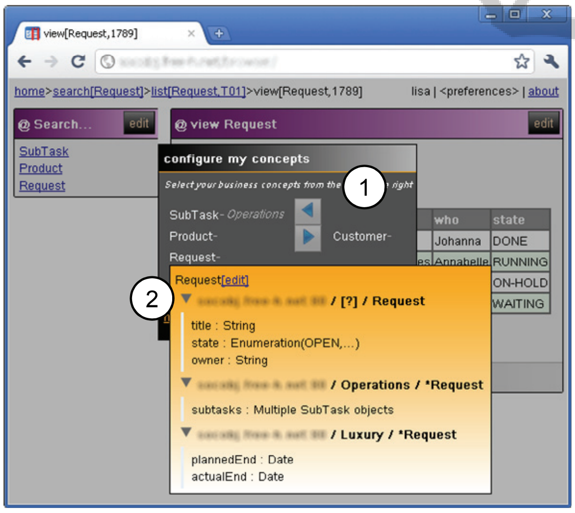
subtasks

SOID	description	who	state
1004	gap analysis	Johanna	DONE
1005	specify missing features	Annabelle	RUNNING
1006	implementation	Fred	ON-HOLD
1007	validation	Ruben	WAITING

plannedEnd October 10,2011
actualEnd

update

Figure 3: Two different users during normal use.


view[Request,1789]

home>search[Request]>list[Request,T01]>view[Request,1789] lisa | <preferences> | about

@ Search... edit

SubTask
Product
Request

@ view Request edit

configure my concepts

Select your business concepts from the ... right

SubTask- *Operations*
Product-
Request-

Customer-

who	state
Johanna	DONE
Annabelle	RUNNING
	ON-HOLD
	WAITING

Request[edit]

[redacted] / [?] / Request

title : String
state : Enumeration(OPEN,...)
owner : String

[redacted] / Operations / *Request

subtasks : Multiple SubTask objects

[redacted] / Luxury / *Request

plannedEnd : Date
actualEnd : Date

Figure 4: A user inspecting his model.

An ideal interface should have the intuitiveness of a spreadsheet, where filling an empty "header" cell transparently creates an extension with the new attribute, with default type and visibility. We believe the presence of actual records makes such example-centric modeling possible.

5.2 Architecture

It may appear natural to host extensions on the same server as their root Elements. However, to fully meet R1 and R2 any Actor must be able to provide his own storage and computing resources for extensions. Otherwise, although independent in functional terms, he is dependent from a physical resource point of view. This constraint dictates a *distributed* architecture, where Perspectives can be hosted on distinct servers and are pulled together at runtime by a client session.

It is important to guarantee that official systems cannot be disrupted or slowed down by extensions hosted on unreliable servers. No organization would accept an architecture with the potential for any unfortunate experiment by an employee to degrade access to central services. This constraint dictates *asynchronous* communication between components, allowing results from a high-reliability official system to be displayed without waiting for the extension results which may arrive later or never.

The prototype implementation is broken down in the following components.

- A central *directory* component, which in a real setting is the enterprise directory server where users and groups would just need to be annotated with references (URLs) to their associated perspectives.
- *Repository* components, which host Perspectives with entity definitions, extensions and associated instances, persisted in a database and exposed through web services.
- On the client-side, the *client session* component communicates with previous components to build a data model at runtime, and a *dynamic user interface* builds simple forms by inspecting this model.

Figure 5 below illustrates the main interactions between the components, at initialization-time and during regular use.

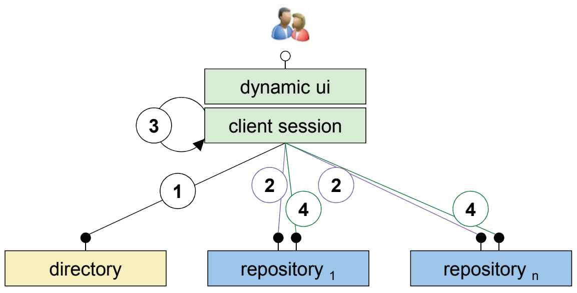


Figure 5: Architecture of the prototype.

In step (1), the client authenticates the user and gets as a reply the full graph of his groups and perspectives. The client then (2) requests all perspectives and the associated Fragment declarations from the various repositories involved. Receiving a Fragment triggers the (3) weaving mechanism which composes Elements. Usage is then similar to any distributed system, i.e. accessing an object triggers several requests (4).

The communication between components is standard REST over HTTP. The protocol has been kept simple in order to enable integration of legacy systems in a perspective-centric landscape through the development of wrappers.

5.3 Limitation

The main conceptual limitation of our first prototype is the focus on data only. Considering the centrality of data in business applications, we think that the results presented in the next section still represent a significant contribution.

6 INITIAL RESULTS

From a technical point of view, the prototype has demonstrated the feasibility of asynchronous runtime composition of a data model, the transparency for end-users during normal use, and end-user update of the data model in their own perspective.

As first proof-of-concept, we have instantiated the prototype with a project tracking use case and a configuration of 3 groups and 5 individuals with different perspectives. The prototype has been able to compose the individual models on the fly, proving the validity of the concepts of Perspective and Fragment.

We have presented this prototype to 8 information system professionals from 6 different industrial and educational organizations. All of them have over 20 years of experience and have witnessed the emergence of numerous shadow applications. Though they did raise some concerns, covered in the discussion section of this paper, their reactions to the proposal varied from fairly positive to enthusiastic. 4 out of 8 subjects have volunteered for evaluating the prototype with real application data.

As a second proof-of-concept, we have instantiated the prototype with the Luxury-like use-case presented in previous sections of this paper. The “Luxury” perspective and its associated “quality” and “operations” perspectives have allowed a unified representation of the three points of view. We were able to walk through use cases of both the Luxury and Fallen shadow applications, and show that technically they would have been avoided with a mature perspective-centric implementation.

The highly dynamic nature of the proposal initially made all interviewed professionals uncomfortable, illustrating the fairly conservative attitude they adopt regarding the architecture of business applications, particularly the data layer. One manager has expressed a desire to restrict the perspective-centric nature of an application to the initial phases of its life, and to “freeze” the model once it has been collaboratively built and validated. This directly contradicted his earlier statements of continuously evolving and conflicting requirements, which he acknowledged. Experimentation with industry datasets is now required to validate our initial results.

7 DISCUSSION

Perspectives represent different, finer and more connected information system grains than applications. We think they allow an information system to evolve organically in a unified and more controlled way than today’s proliferation of shadow applications, without sacrificing the business units’ ownership of their specific application elements. The reactivity and autonomy their mission demands is thus preserved.

7.1 Towards Social Information Systems

A perspective-centric application architecture represents a major shift of responsibilities from IT departments towards the community of users, not unlike the freedom spreadsheets have provided (Nardi and Miller, 1990). An IT department’s main responsibility would be to provide the platform on which anyone (the IT department itself, but also business organizations and individual employees) could contribute elements in their area of expertise. We think this could leverage the collective intelligence (Surowiecki, 2004) and energy of employees to collaboratively build and maintain the corporate information system, in a form of internal crowd-sourcing.

Considering today’s mostly feudal management of information systems, this is a fairly disruptive proposal. Indeed, during our interviews most subjects have raised the concern that it could result in chaos. This concern typically takes the official applications as a reference, while in our opinion it only represents the tip of the information system iceberg. When including all shadow applications in the picture, information systems today can already not guarantee the overall consistency, and rely upon humans to keep the whole together. However, as one architect interviewed observed, the chaos is often feared to be in core business attributes. But these are often the best-understood and least controversial of the data elements; uncertainty is greatest on the highly domain-specific attributes. By properly segregating these into the correct perspectives, overall uncertainty may actually be reduced.

Collecting all shadow application data in a unified infrastructure may seem to aggravate inconsistency, but in reality it just reveals the present state. We think a unified infrastructure would provide additional leverage to the previously mentioned human factor in at least two ways.

In the consumer-space, "social" mechanisms like tagging, rating, voting, and targeted sharing have proven effective in organizing huge repositories of consumer-contributed data (Surowiecki, 2008). In a business environment, users could organize application elements through similar mechanisms. We think dealing with authenticated professionals is an even more beneficial setting than the consumer space for social technologies to apply, and envision *social information systems* where elements are contributed from the bottom up, shared with other Actors, ranked and improved through social feedback mechanisms and eventually gradually "promoted" to more central perspectives.

This could result in the *democratic* (or *meritocratic*) evolution of a corporation's application landscape, a generalization of today's frequently requested transfer of shadow applications from business units to IT departments (Handel and Poltrock, 2010).

As opposed to today's situation where shadow applications are mostly disconnected from their parent applications and extremely heterogeneous in their implementation, a unified architecture would make the continuous evolution and divergence *observable*. Indicators could be envisioned (number of extensions, number of unimports…) and dashboards built to monitor application evolution. Pattern-matching techniques could be used to automatically detect convergence opportunities (Ahmadi, Jazayeri, Lelli and Nesic, 2008; Sabatzadeh, Finkelstein and Goedicke, 2010) and notify the owners of the candidate elements, fostering convergence discussions.

7.2 Impact on Collaboration

Although the goal of the proposed architecture is to make evolution a continuous process, introduction of significant chunks still require traditional projects.

From a functional point of view, the painful and hazardous process of elaborating the union of divergent requirements could be replaced by the identification of the intersection, containing only the elements all stakeholders agree on, and then spawn smaller groups to discuss the next level of detail, thus reducing the risk of conflict and communication overhead. We think Perspectives would thus contain the various layers of *boundary objects* (Star, 1990) around which people collaborate.

From a technical point of view, private spaces could help in integrating running development projects with live production environments, facilitating continuous integration and delivery (Fowler, 2010a). Boundaries between mockup, prototype, beta and production environments could be smoothened and concurrent development made easier, as well as quick experimentation encouraged.

7.3 Evaluation in the Real World

One of the challenges of this work is to find suitable ways to evaluate the underlying concepts of social information systems. A standard approach would be to deploy this with a small group of users, and study its usage. However, if it were deployed in this fashion, it would become just one more shadow application, and many of the benefits of a perspective-centric system would be lost. On the other hand, this approach is new and unfamiliar enough to both potential users and IT organizations that a major implementation would be difficult to accomplish. As illustrated by the aforementioned discomfort of the IT professionals, this requires a significant shift in thinking by IT and line-of-business managers about how crucial data is stored and managed. A successful perspective-centric system requires not only technological sophistication, but also a degree of organizational change that is not always present (the "c-words").

7.4 Challenges and Further Work

A real deployment of such a social architecture would almost certainly exhibit a high degree of coupling of its elements, making the system vulnerable to the evolution of central elements. However, since all dependencies are explicit, *evolution policies* could be defined. For example, if a high-level perspective deletes an element, it could be marked as orphan and be proposed to adoption to owners of perspectives which import or extend it.

We think a significant number of common business application features can be implemented in a generic way in the form of *functional aspects* (Filman, Elrad, Clarke and Akşit, 2008) to be *applied* by an end-user while building his model. For example, if a particular attribute demands traceability this could be a single checkbox on the model's form, a simple boolean annotation on the model itself, and could tell a repository to produce history records with timestamp, user, and previous value. We are working on more complex aspects like lifecycle management and authorization.

The manipulation of model and instances through the same interface presents both the opportunity to leverage contributions from people without modeling skills and the risk to confuse them. Beyond the prototype's naïve forms for model manipulation, we consider usability for contributors with a broad spectrum of software skills a challenge. For contributors with software engineering skills, the development of robust application code on top of a dynamic foundation is not trivial, and needs appropriate programming language bindings.

Other challenges are not new but rather inherited from the present situation. As an example, a user could define an extension concatenating two attributes, and export this extension to colleagues who do not have permission to see the initial data. This is similar to what happens when people extract confidential data in today's shadow applications, breaking the initial authorization mechanism. A perspective-centric system would actually improve on this situation; by having a complete view of all the attributes, a system would be able to detect and warn about possible permission violations.

At a higher level, perspective-centric architectures present a number of interesting challenges, like monitoring and convergence mechanisms, and adapting the consumer-space social recommendation mechanisms to application elements in a business environment.

8 RELATED WORK

We consider the work presented in this paper a novel combination of existing approaches. Shadow applications are a widely known but widely accepted problem. They are frequently mentioned when studying information system agility (Desouza, 2007) or dissatisfaction with business applications (Hoyer and Stanoevska-Slabena, 2008), but not always considered as a problem (Handel and Poltrock, 2010).

Situational applications are enterprise applications built on-the-fly by business units to solve a specific business problem (Markl et al., 2008), and can be considered a superset of shadow applications. Situational applications have attracted recent interest from enterprise mashup researchers (Hoyer and Fischer, 2008) who aim at allowing end users to integrate and combine services, data and other content (Bitzer and Schumann, 2009) to bridge the IT/business gap. Mashups can be interpreted as an evolution of service-oriented architectures (Watt, 2007), which expose business functionality as standard and composable services.

Mashups are part of the broader topic of end-user development (Nestler, 2008); (Sutcliffe, 2005), which advocates the empowerment of end-users to implement their own specific requirements, and has intensively studied spreadsheets (Nardi and Miller, 1990); (Spahn and Wulf, 2009) and more recently collaborative and social aspects in enterprise settings (Ahmadi et al., 2009).

Model-Driven Engineering (Schmidt, 2006) elevates the level of abstraction at which software is developed, turning models into central and productive artifacts, with a specific models@runtime branch focusing on model interpretation. The Software Language Engineering (Kleppe, 2008) and Domain-Specific Languages (Fowler, 2010a) domains, related to MDE by the heavy reliance on meta-models, focus on domain expert involvement in software development and configuration through specific textual representations.

The Component-Based Software Engineering (McIlroy, 1968) community is actively researching robust dynamic systems, where components can appear and disappear during execution. It provides foundation concepts and technologies for making a social application cope with dynamic elements and services of variable reliability.

Linked Data (Bizer, Heath, and Berners-Lee, 2009) integrates distributed, loosely coupled and independently managed repositories of persistent entities, but targets an internet-wide database and mostly-read access.

Social Software Engineering focuses on the understanding of the human and social aspects of software engineering. It covers both the social aspects in the software engineering process and the engineering of social software (Ahmadi et al., 2008). In the Requirements Engineering domain, Lohmann et al. (2009) propose to apply social mechanisms like voting and commenting. Studies on ViewPoints (Sabetzadeh et al., 2010) have focused on capturing divergent concerns but aim at reconciling these at the specification and design level.

The tailoring of enterprise systems, from simple configuration to the modification of commercial code, is a topic of sufficient complexity for (Brehm et al., 2001) to propose a typology. Recent interest in cloud computing has yielded research in multi-tenancy (Jansen et al., 2010), a way to configure the same software installation for various isolated organizations.

9 CONCLUSIONS

In this paper we have presented an alternative architecture for business applications designed to reduce shadow application proliferation. We have described the main characteristics of shadow applications, the causes of their emergence, and have proposed an architecture principle to defuse this phenomenon based on an isolation mechanism we call *perspectives*. We have presented our prototype, our first results on real-life use cases and the encouraging feedback it has received.

We have discussed our broader vision of a social information system leveraging the collective intelligence of an organization's employees, and the possibility of democratic evolution through the use of social mechanisms.

We have no silver bullet claim, rather a potentially interesting paradigm worth exploring. We have no revolution claim either, merely an original combination of existing approaches and a generalization of business application configuration mechanisms. This is enabled by continuously growing processing power versus fairly stable core requirements of business applications, a better understanding of distributed systems, and recent social technologies.

ACKNOWLEDGEMENTS

This work has been funded by Nano-2012 grant MoDeSI. The authors would like to thank the participating interviewees for their time and helpful comments.

REFERENCES

Ackerman, M. S., 2000. The Intellectual Challenge of CSCW: The Gap Between Social Requirements and Technical Feasibility, *Human-Computer Interaction*, Volume 15

Ahmadi, N., Jazayeri, M., Lelli, F. and Nesic, S., 2008. A Survey of Social Software Engineering, *IEEE/ACM ASE - Workshops*

Ahmadi, N., Jazayeri, M., Lelli, F. and Repenning, A., 2009. Towards the Web of Applications: Incorporating End User Programming into the Web 2.0 Communities, *Proc SoSEA 2009*, ACM

Bitzer, S. and Schumann, M., 2009. Mashups : An Approach to Overcoming the Business/IT Gap in Service-Oriented Architectures, *Value Creation in e-Business Management*, ISBN 978-3-642-03131-1

Bizer, C., Heath, T. and Berners-Lee, T., 2009. Linked Data - The Story So Far, *International Journal on Semantic Web and Information Systems*

Brehm, L., Heinzl, A. and Markus, M. L., 2001. Tailoring ERP Systems: A Spectrum of Choices and their Implications, *Proc HICSS*, IEEE

Desouza, K. C., ed., 2007. *Agile Information Systems : Conceptualization, Construction and Management*, ISBN 978-0-7506-8235-0

Filman, R. E., Elrad, T., Clarke, S. and Akşit, M., 2008. *Aspect-Oriented Software Development*, ISBN 0-321-21976-7, Addison-Wesley, 2008

Fowler, M., 2010. *Domain-Specific Languages*, ISBN 978-0-321-71294-3, Addison-Wesley

Fowler, M. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, ISBN 978-0-321-60191-9, Addison Wesley, 2010

Gotts, I., 2010. A New Cloud: The Stealth Cloud?, *http://www.cio.com/article/630164*, October 2010

Handel, M. and Poltrock, S., 2010. Working Around Official Applications : Experiences from a Large Engineering Project, *Proc CSCW,* ACM Press

Hoyer, V. and Fischer, M., 2008. Market Overview of Enterprise Mashup Tools, *Proc ICSOC*, Springer Verlag

Hordijk, W. and Wieringa, R., 2010 Rationality of Cross-System Data Duplication: A Case Study, *Proc CAiSE*, Springer Verlag

Hoyer, V. and Stanoevska-Slabena, K., 2008. The Changing Role of IT Departments in Enterprise Mashup Environments, *Proc SOC*, Springer-Verlag

Jansen, S., Houben, G-J. and Brinkkemper, S., 2010. Customization Realization in Multi-tenant Web Applications: Case Studies from the Library Sector, *Proc ICWE*, Springer-Verlag

Kleppe, A., 2008. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*, ISBN 9780321553454, Addison-Wesley

Kling, R., 1991. Cooperation, Coordination and Control in Computer-Supported Work, *Communications of the ACM*, Volume 34 Issue 12

Lohmann, S., Dietzold, S., Heim, P., Heino, N., 2009. A Web Platform for Social Requirements Engineering, *Software Engineering Workshops 2009*

Markl, V., Altinel, M., Simmen, D. and Singh, A., 2008. Data Mashups for Situational Applications, *Proc MBSDI 2008*, Springer-Verlag

McIlroy, D., 1968. Mass-Produced Software Components, *Software Engineering, Report on a conference*

sponsored by the NATO Science Committee
Nardi, B. A. and Miller, J. R., 1990. An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development, *Proc CSCW 1990*, ACM Press
Nestler, T., 2008. Towards a Mashup-driven End-User Programming of SOA-based Applications, *Proc iiWAS 2008*, ACM Press
Newell, S., Wagner, E. L. and David, G., 2007. Clumsy Information Systems: A Critical Review of Enterprise Systems, *Agile Information Systems*, Elsevier
Sabetzadeh, M., Finkelstein, A. and Goedicke, M., 2010. ViewPoints, *Encyclopedia of Software Engineering*, P. Laplante, Ed.
Schmidt, D. C., 2006. Model-Driven Engineering, *IEEE Computer Vol 39*
Spahn, M. and Wulf, V., 2009. End-User Development for Individualized Information Management: Analysis of Problem Domains and Solution Approaches, *Proc ICEIS 2009*, Springer Verlag
Star, S. L., 1990. The Structure of Ill-Structured Solutions : Boundary Objects and Heterogeneous Problem Solving, *Distributed artificial intelligence*, Vol. 2, Morgan Kaufmann
Surowiecki, J., 2004. The Wisdom of Crowds, ISBN 978-0385503860
Sutcliffe, A., 2005. Evaluating the costs and benefits of end-user development, *ACM SIGSOFT Software Engineering Notes*, Vol 30
Watt, S., 2007. Mashups - The evolution of the SOA: Situational applications and the Mashup ecosystem, *http://ibm.com/developerworks/webservices/library/,* Nov 2007
Zarnekow, R, Brenner, W. and Pilgram, U., 2006. *Integrated Information Management: Applying Successful Industrial Concepts in IT*, ISBN 978-3540323068