Systèmes d'information sociaux

Marc Quast

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **INFORMATIQUE**

Arrêté ministériel : 7 août 2006

Présentée par

Marc QUAST

Thèse dirigée par **Jacky ESTUBLIER** et **Jean-Marie FAVRE**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale MSTII**

Social Information Systems

Thèse soutenue publiquement le **24 Octobre 2012**,
devant le jury composé de :

Mme. Joëlle COUTAZ
Professeur, Université de Grenoble, Président

Mme. Tamara BABAIAN
Associate Professor, Bentley University, United States of America, Rapporteur

M. Mehdi JAZAYERI
Professor, University of Lugano, Switzerland, Rapporteur

M. Pierre-Alain MULLER
Professeur, Université de Haute-Alsace, Examinateur

Université Joseph Fourier / Université Pierre Mendès France /
Université Stendhal / Université de Savoie / Grenoble INP

RESUME

Les systèmes d'information d'entreprise actuels s'articulent autour d'applications centrales lourdes, qui ne fournissent pas l'agilité nécessaire pour survivre dans un environnement économique hautement concurrentiel. De nombreux acteurs (unités, individus, équipes et communautés) doivent introduire leurs propres applications pour pallier à ces limitations, avec pour résultat un système d'information fragmenté, incohérent et impossible à gouverner.

Cette étude propose un paradigme d'architecture d'entreprise alternatif, qui s'appuie sur une décomposition plus fine du système d'information et une distribution différente des responsabilités. Il permet à tout acteur de contribuer au système d'information en introduisant des fragments, privés ou partagés avec d'autres acteurs, qui peuvent ensuite être composés pour former des applications dédiées à un profil. Les récents mécanismes de l'informatique sociale sont proposés pour gérer les volumes potentiels importants de fragments émergeant de la communauté d'employés.

L'objectif des systèmes d'informations sociaux est à la fois d'améliorer la cohérence et la gouvernabilité du système d'information de l'entreprise et d'exploiter l'intelligence et l'énergie collective de l'entreprise à des fins d'agilité métier maximale.

Mots-clés

Systèmes d'Information Entreprise, Applications, Agilité, Logiciels Sociaux,
Ingénierie Dirigée par les Modèles, End-User Development, Composition de Logiciel.

ABSTRACT

Present enterprise information systems are centered on heavy corporate applications, which cannot and indeed do not provide the agility required to survive in todays' competitive business landscape. Actors (business units, individuals, teams and communities) must introduce their own applications to work around these limitations, resulting in a fragmented, inconsistent and ungovernable information system.

This thesis proposes an alternative enterprise architecture paradigm based upon a finer-grained decomposition of information systems and a different distribution of responsibilities. It empowers all actors to contribute fragments to the information system, private or shared with other actors, which can then be composed to form profile-specific applications. Consumer-space social mechanisms are proposed to manage the potentially huge resulting numbers of fragments emerging from the employee community.

The aim of social information systems is both to improve the overall consistency and governability of the enterprise information system and to leverage the collective intelligence and energy of the corporation towards maximum business agility.

Keywords

Enterprise Information Systems, Business Applications, Agility, Social Software,
Model-Driven Engineering, End-User Development, Software Composition.

Contents

Index of Figures

1. Introduction

1.1. Motivation

Present enterprise information systems are centered on heavy corporate applications, which cannot and indeed do not provide the agility required to survive in todays' competitive business landscape. Dissatisfaction with information systems is widespread.

Due to the strategic importance of information systems, corporations typically don't volunteer evidence of their shortcomings. As a notable exception, The Boeing Company has recently published an experience report describing the alarming numbers of shadow applications which various actors (business units, individuals, teams and communities) must introduce to work around the limitations of the central information system. Due to both our own experience in manufacturing industries and discussions with professionals from various domains (banking, telecommunications, healthcare and government), we consider this a global phenomenon.

The result is a heavily fragmented information system, where consistency and governability have been sacrificed to achieve a reasonable level of business agility. Our opinion is that with present software architectures, no matter how carefully business applications are crafted, over time they will spawn shadow applications whenever resourceful actors have urgent unsatisfied needs, solving a local problem but aggravating the overall situation.

Our hypothesis is that a different information system architecture principle is both necessary and possible.

1.2. Main Contributions of this Thesis

This thesis proposes an alternative enterprise architecture paradigm based upon a fundamentally different distribution of responsibilities, empowering all actors to adapt business applications themselves without introducing new applications and without impact on other actors.

This is achieved by the decomposition of business applications into smaller fragments, enabling multiple subjective representations of the corporate reality which defuses the bottleneck of requirements negotiation. The ownership of such fragments can then be distributed, allowing all actors to introduce the fragments they need, eliminating implementation bottlenecks as well.

We describe a social information system, where all actors contribute and share business application fragments, applying social technologies to organize, rank and propagate high numbers of emergent fragments, which can gradually be "promoted" to more central perspectives, enabling the collaborative design and meritocratic evolution of the corporate information system.

The potential benefits of social information systems are to improve the overall consistency and governability of the enterprise information system and to leverage the collective intelligence and energy of the corporation towards maximum business agility.

1.3. Thesis Structure

Chapter 2 provides a critical observation of the present state of information systems in big corporations, including the description of the real-life use cases which will serve as the running example of this study. **Chapter 3** generalizes these observations in the form of a set of high-level requirements for an agile information system.

Chapter 4 then evaluates the two dominant information system architecture paradigms with respect to these requirements, highlighting strengths and weaknesses and demonstrating the need for an alternative enterprise architecture principle.

Chapter 5 proposes such a principle, based upon a fundamentally different granularity of business applications and an alternative distribution of information system responsibilities. It discusses our vision of a social information system leveraging the collective intelligence of an organization's employees, and the possibility of its collaborative evolution.

Chapter 6 refines this proposal by describing a possible architecture, based upon a set of components for which it provides a high level design and brief discussion, and **chapter 7** presents a prototype implementation of a subset of these components.

The prototype implementation has allowed a number of experiments on both fictional and real-life use-cases, which **chapter 8** presents along with an evaluation of the concepts, architecture and implementation versus our initial requirements for business agility.

Chapter 9 concludes the study. An index of the terms we define or use is provided at the end of the document.

1.4. Notation

The UML notation [1] is used in the majority of diagrams of this document. When describing technical artifacts, the standard notation is used. When merely disambiguating real-world terms, the pseudo hand-written notation below is used.

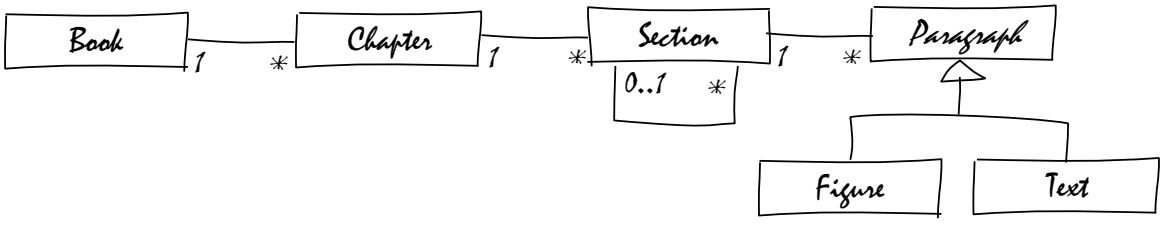


Figure 1-1. Hand-written notation used for disambiguation of common terms

2. Enterprise Information Systems

This chapter presents the context of our study, namely the information systems of big corporations. We describe the need for business agility and organizational complexity, and both the importance and high-level structure of business applications. We present shadow applications, which we describe in detail. We analyze their benefits, drawbacks and of the causes of their emergence, which together will provide the foundation for the research hypotheses expressed in the next chapter.

Terminology

When describing organizations and their employees, we will use the terms *actor*, *individual* and *group* according to the following model.

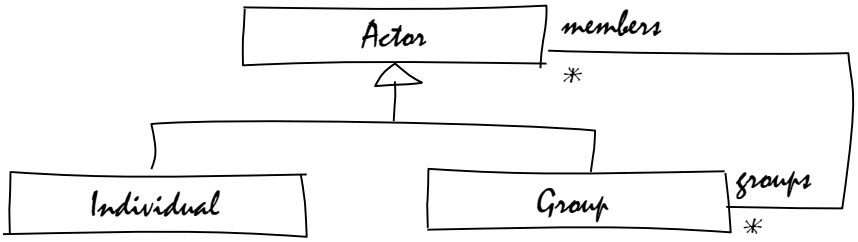


Figure 2-1. Disambiguation of the terms actor, individual and group (class diagram)

It is important to notice that group is recursive, which allows to represent organization structures of varying depths. When it is necessary to distinguish between typical groups, we will use the terms *corporation*, *business-unit* and *department* to refer to three common organization levels described and illustrated below.

- The term *corporation* refers to the top of the organization, with a typical size between 10 000 and 100 000 employees (individuals)
- The term *business-unit* indicates groups close to the top, with typical sizes between 1 000 and 10 000 employees
- The term *department* designates the next level of organization, between 100 and 1000 employees

- We will use the terms *team* to refer to the lowest level of organization, and both *project* and *community* to designate cross-organizational groups, not represented in the diagram below and varying in size from a few to thousands of employees

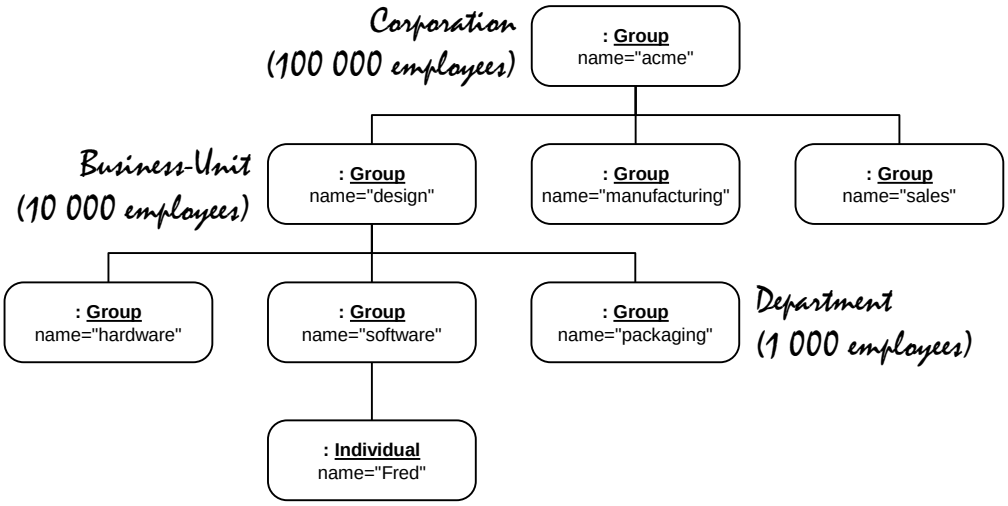


Figure 2-2. Corporation, Business-Units and Departments (instance diagram)

It is also important to highlight that in the Figure 2-1 class diagram, an actor can be a member of multiple groups, which allows to represent not only a traditional hierarchy (i.e. a tree), but the multiple dimensions (i.e. a graph) necessary to represent complex organizations as depicted in Figure 2-3.

2.1. The Need for Business Agility

Today's business landscape is highly competitive and dynamic. Markets are global, international competition is fierce, change happens more rapidly and more often in the form of discontinuous upheavals rather than incremental changes [2]. The very notion of the classical enterprise is evolving, and alternative forms of organization, particularly flexible networks, are emerging [3]. In such an environment, *business agility* is a key element for corporations to survive and be successful [4].

There is no consensus yet as to what business agility is. A number of definitions can be found in the literature [5, 6, 7, 8, 9]. In an attempt to disambiguate flexibility and agility, [10] proposes to define flexibility as *"a predetermined response to a predictable change"*, and agility as *"an innovative response to an unpredictable change",* focusing flexibility on *"single systems with medium frequency of change"* and agility on *"groups of systems with high change rates"*. [11] summarizes the various views on agility as *"a way to cope with highly uncertain external and internal changes"*.

Business agility includes both *sensing* capabilities (detection and anticipation) and *responding* capabilities (physical ability to act rapidly and with relative ease) [9]. The present study focuses on response capabilities, and we adopt the following restrictive definition for the remainder of this document.

definition **Business Agility**
Ability of a corporation to adapt rapidly and efficiently (i.e. at low cost) to unpredictable changes in its environment.

Small start-up companies are agile by construction. They consist of a small group of individuals, focused on one particular business opportunity. Big established corporations however suffer from inertia almost by definition. In order to have large groups of individuals work together efficiently, corporations aim at the right balance between the following two organization models.

- In a *centralized* model, a strong hierarchical structure ensures consistency, through clear decision and communication channels and emphasis on well-defined reproducible processes, typically at the cost of heavy bureaucracy and bottlenecks.
- In a *decentralized* model, a network of independent groups focus on their specific goals, which ensures high autonomy[1] and high reactivity when facing business opportunities or threats, at the cost of a complex global organization and duplications of efforts.

A common compromise is to centralize support functions for cost-effectiveness and to decentralize core business functions for optimal agility.

Regardless of the balance between centralization and decentralization, *information technologies* are vital [12, 4] and a key factor for achieving business agility. Within the broad field of business information technologies, our study focuses on *business applications*. Before we present business applications in section 2.3, it is important to highlight the complexity of big corporations.

2.2. Organizational Complexity

The organization of big multinational corporations is a research topic in its own right. For the purpose of our study, we highlight the existence of multiple organizational dimensions, as illustrated by the figure below showing an individual Maria belonging to multiple groups across 5 orthogonal dimensions, most dimensions themselves hierarchical. The organization of real corporations is more complex, both wider and deeper.

[1] In extreme situations this can mimic startups (“intrapreneurship”)

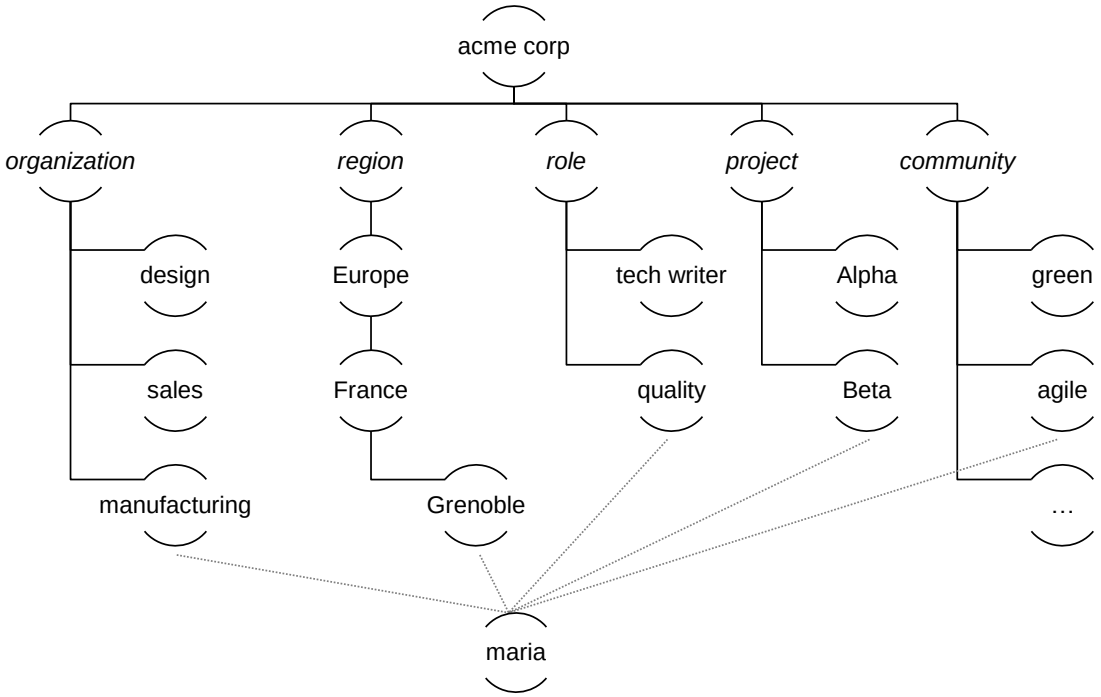


Figure 2-3. An example partial organization structure with 5 dimensions

The dotted lines represent the individuals' membership of certain groups[2]. We will refer to the combination of group memberships as a *profile*.

definition **Profile**
Combination of all groups an actor is a member of.

In the example above, Maria's profile is a person playing a quality role in a manufacturing organization on a site in Grenoble, France, who also participates in project Beta and is a member of a workgroup on agility. In big corporations, there are a high number of possible profiles, potentially equal to the number of its knowledge workers, i.e. tens of thousands.

2.3. Information Systems and Business Applications

There is no general agreement on the definition and goals of enterprise information systems [13]. In this document, we will use the term *information system* to refer to the set of all business applications used in a given corporation, excluding aspects like business processes and people.

definition **Information System**
Set of all business applications used in a given corporation.

[2] This is a simplification of reality in two respects. First, memberships can be factorized at various group levels, resulting in a significantly more complex graph. Second, memberships can be contextual, a common example being an individual with a "quality" role which extends beyond the organization or site he belongs to. This document adopts a simplified view for the purpose of clarity, but we think the results of our study will prove even more relevant with the real complexity in mind.

The term *application* itself is heavily overloaded, and we will use in the broad sense in this document. We have published a disambiguation in [14], included in appendix A. The class diagram below illustrates this definition, and introduces the associated term *end user*, which we define as an individual using an application to perform his work [15].

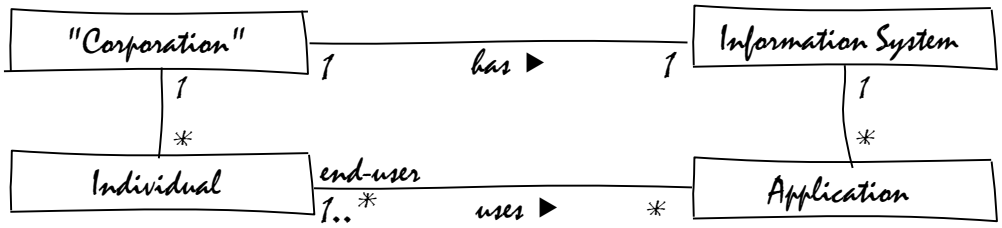


Figure 2-4. Information System and Applications

2.3.1. Business Application Elements

Business applications provide a way for individuals to interact with persistent business records. The work of knowledge workers depends on a high extent on manipulating such business data through a user interface [16]. Interactions mainly involve querying, creating, updating and deleting such records, a set of features often referred to as CRUD, i.e. create, retrieve, update and delete. Besides displaying and updating various attributes of business records, applications allow to invoke business functions of low to moderate complexity. It is common to separate these concerns into the tiers presented in the figure below.

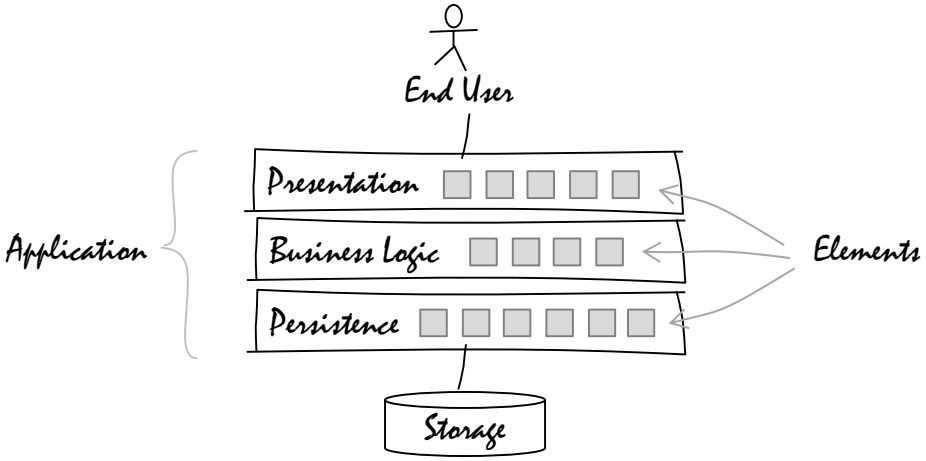


Figure 2-5. Typical tiers of modern business applications

In this section we provide a high-level view of the main elements of these three tiers, and highlight that they have remained relatively constant from the point of view of the end user.

2.3.2. Persistence Elements

The responsibility of the persistence layer is to encapsulate the permanent storage of structured business objects, and to isolate the upper tiers from persistence-related specifics like data structures, indexes and data partitioning. Technologies have evolved from flat files to various generations of database management systems [17]. Business records have evolved into business objects in the 1990s [18] and structured documents [19] later. Regardless of the

underlying technologies, from a user perspective the persistence tier roughly conforms to the following meta-model.

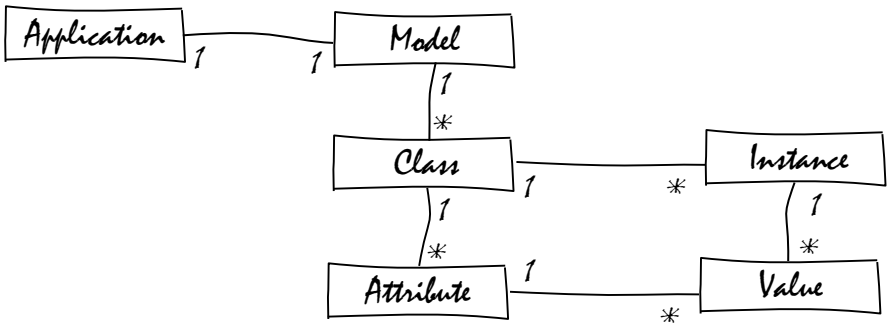


Figure 2-6. High-level meta-model of persistence elements

In enterprise software, the total number of classes can vary from under 10 to several thousands in ERP systems. While we can observe increasing complexity, i.e. number of model elements, and increasing volumes, i.e. numbers of instances [20], we can consider that the core concepts presented in the above figure have remained fairly constant over the history of business applications.

Relational database systems are the dominant technology in the persistence tier, and it is easy to establish the parallel between Class, Attribute and Instance from the meta-model above with respectively Table, Column and Row in the relational database space.

Figure 2-6 also introduces the term *model* in the restrictive but common meaning encompassing the set of business classes underlying an application.

2.3.3. Business Logic Elements

The business logic tier or middle tier has emerged as a response to the fat client syndrome, which tangled business concerns with presentation concerns [21]. It factorizes access to persistent business objects by exposing a *service* interface. Besides the aforementioned CRUD operations, the business logic tier provides higher-level domain-specific services across multiple business objects.

A classic example of such a service is the transfer of money from one account to another. This routine operation involves a number of business rules (like checking that after withdrawal from the source account, the balance is not below its minimum) and can trigger workflows (approval of the transfer depending on the amount), notifications, etc. A service typically ensures that the user has the proper authorizations to perform the operation.

The purpose of factorizing such elements in the middle tier is to capture the business process and rules for reuse by multiple clients; in the case of our money transfer example, the bank clerk's application, the ATM application or the online banking application used by the account holder at home are all clients using the same service. Consistent processing is thus guaranteed independently from the number of presentation elements invoking the operation.

2.3.4. Presentation Elements

The first figure shows a screenshot of a text-based business application as common in the 1970s, based on 80-columns text terminals and function-key based user interaction. Even today, such applications are still commonplace in manufacturing, banking, retail and administration environments alike.

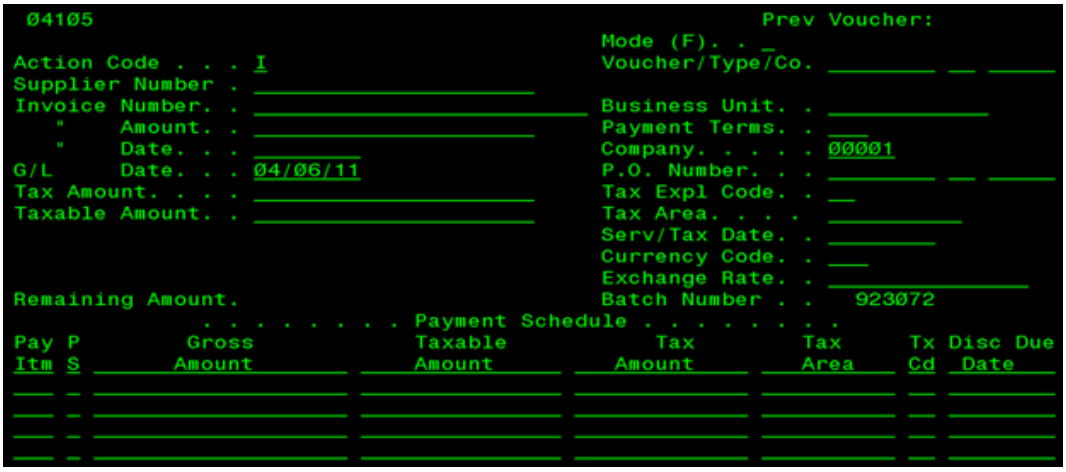


Figure 2-7. Screenshot of a text-based business application

Text-based business applications revolve around successive *forms*. Forms are composed of distinct sections, displaying fields with labels and values, in stacked or tabular format, and a set of actions. Navigation through the form sequence is a particular case of action, and we can consider menus a particular case of form. The class diagram below provides a high-level meta-model of the core user interface elements of text-based business applications.

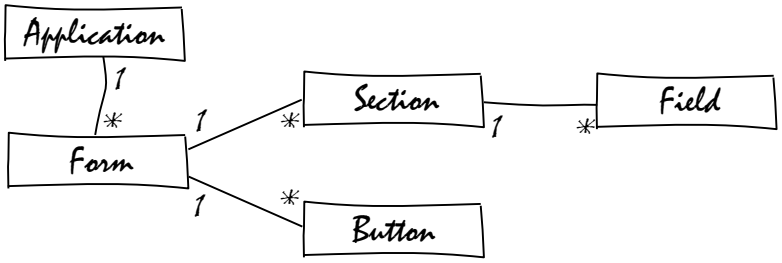


Figure 2-8. High-level meta-model of text-based business application user interface elements

In the mid-1990s, *graphical* user interfaces became common, used in windowing environments with pointer devices. Besides minor changes like clicking on buttons instead of pressing function keys, graphical business applications reproduce the original text-based interaction. The figure below shows a screenshot of a graphical business application, illustrating the similarity with their text-based predecessors and thus the reasonable conformance to the meta-model above.

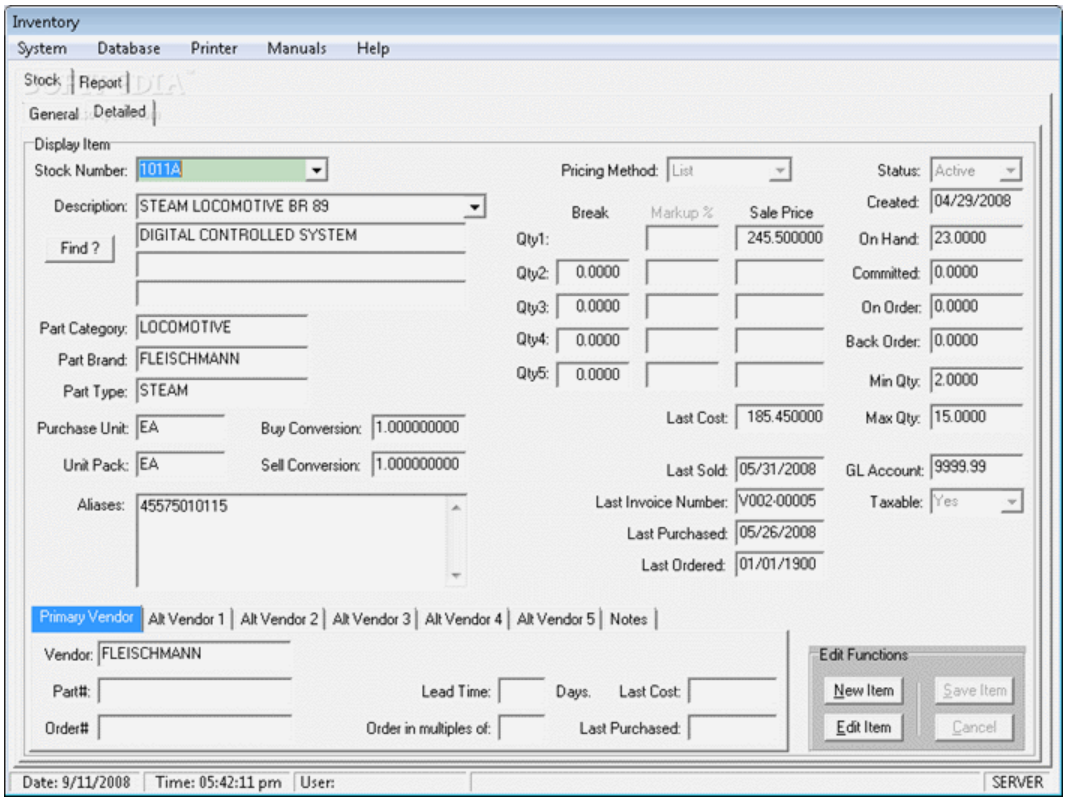


Figure 2-9. Screenshot of a first-generation graphical business application

Starting in 2000, mainly in order to solve deployment and upgrade problems, the norm has gradually become to provide *web-based* interfaces. Whether in the early passive HTML format or in one of the present RIA[3] technologies, web-based applications again retain the familiar form-centric interaction model. Such applications are the state-of-the-practice today, as illustrated by the figure below showing Oracle's CEO Larry Ellison during his keynote speech at the Oracle World 2011 conference, announcing Oracle's next generation of business applications, the result of 6 years of engineering efforts.

[3] Rich Internet Application, via technologies like Javascript, Adobe/Flash or Microsoft/Silverlight.

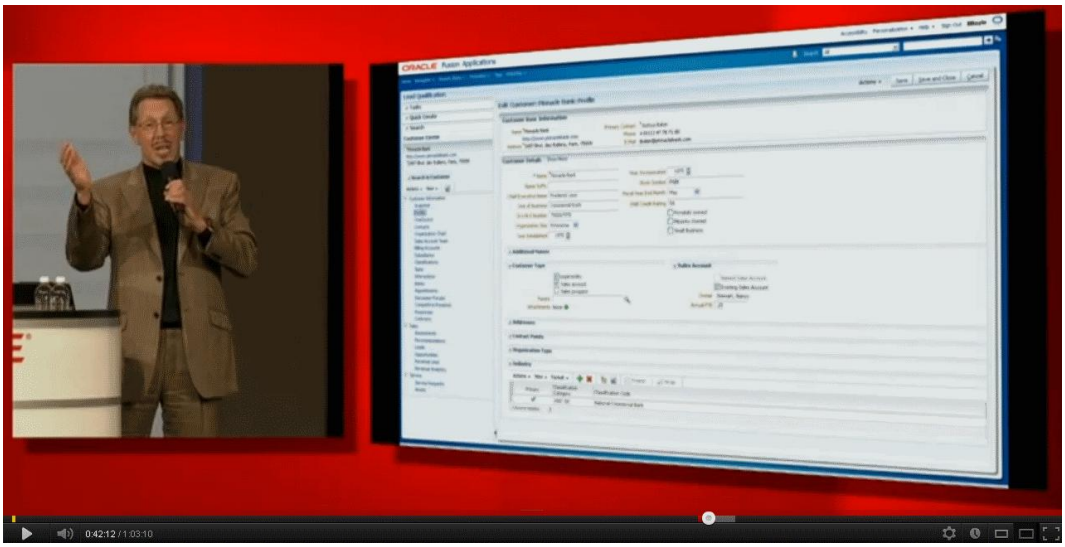


Figure 2-10. Oracle's CEO announcing their new generation of business applications in 2011 [22]

We will thus assume in our study that the present form-based user interfaces of business applications are satisfactory and will remain so. The emerging trend of mobile business computing, besides introducing a few new interaction modes, is unlikely to fundamentally change business forms. The associated shrinking screen real-estate may actually urge the profession to revert back to simpler interfaces with fewer elements.

2.3.5. Summary

Through several decades of technological evolutions, including the internet and mobile communication breakthroughs, both the inner and outer core characteristics of business applications have remained fairly constant [23]. They revolve around persistent structured data with complex data models and high numbers of instances. Users invoke fairly simple operations through forms.

The diagram below summarizes these constant concepts of business applications, and introduces the abstract class *Element* to designate them. We will reuse the term element in the remainder of this document with this rough meaning, i.e. any piece of a business application which is perceived by the end user.

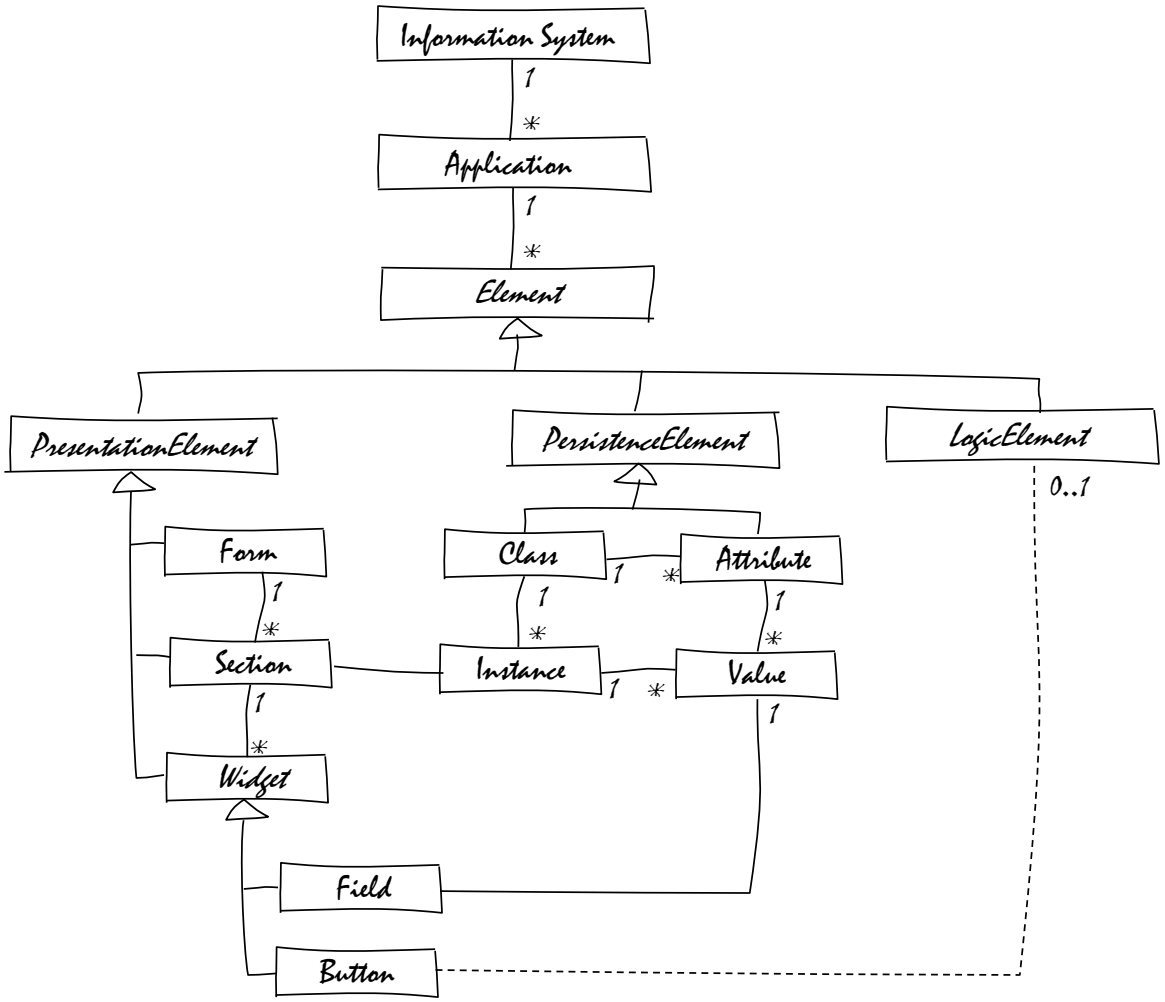


Figure 2-11. Core Elements of business applications

Due to both the complexity of generalizing the business logic tier and the data-centric nature of business applications, the remainder of this study will not further elaborate LogicElements and consider the create-read-update-delete operations as representative.

In the next section, we discuss how Information Systems and Business Applications both contribute to and hinder business agility.

2.4. The Problem: Information Systems are not Agile

A common enterprise IT strategy is labeled "buy before build"[4]. Corporations adopt commercial enterprise application packages like SAP, Oracle Applications or Baan, with the expectation of both more robust and less expensive systems compared to in-house developments, faster scaling and benefitting from "industry best practices" embedded in the tool, theoretical benefits which we will not question in this study. Corporations adopting such enterprise applications must configure and customize them, integrate them within their application landscape, and more often than not modify their source code in order to meet vital

[4] An extended version of this principle is « reuse before buy ; buy before build »

business requirements [24, 25]. Commercial enterprise applications thus don't fit the traditional distinction between custom-built and COTS[5] applications [26]. We will refer to this activity as *tailoring*.

definition **Tailoring**
Configuration, customization or modification performed to adapt business application software to specific business requirements.

Tailoring is often considered critical to gain a competitive advantage over competitors, whether these are using the same application [27] or not [28]. Keeping or augmenting this competitive advantage makes tailoring a key activity for business agility [16].

2.4.1. Corporate IT and the Rise of Shadow IT

In order to achieve business agility, groups need to swiftly adapt to changes in their environment, which more often than not has impacts on the information system. Actors thus frequently need to change business applications. In a typical company however, applications are owned by a central IT department[6]. We will call this department the *corporate IT department*, and use both the prefixes *corporate* and *official* as adjectives for everything under this departments' control.

definition **Corporate IT Department**
Department centralizing IT activities for a corporation.

A corporate IT department presents the usual benefits of centralization, mainly saving costs and enforcing standards. They also have the usual drawbacks, i.e. low reactivity, low service transparency, poor support and insufficient understanding of specific requirements [29], aggravated by the common practice of aggressive cost cuttings which precludes improving the quality of their service.

As a result, the business applications provided by corporate IT departments are a widespread source of frustration [30]. They do not allow sufficient nuance, are not socially flexible, and do not allow sufficient ambiguity to adequately support day-to-day requirements [31]. Business-IT alignment [32] has become a field actively studied by both researchers and practitioners [33, 34, 29, 35].

In order to achieve their goals, business units cannot and indeed do not accept the poor service provided by their IT departments, and tend to build up independent IT resources to suit their specific or urgent requirements [36] in order to meet their objectives. This phenomenon is widely observed [29], and goes by various names. We will adopt the term *shadow IT*.

definition **Shadow IT Activity**
IT activity performed by an actor who is not a member of the corporate IT department.

[5] Commercial-Off-The-Shelf
[6] Outsourcing [176] is an external form of centralization.

Shadow IT activities are thus performed by actors while it is not (a) their or (b) their organizations' primary mission. Shadow IT actors can be (a) isolated individuals with sufficient interest in IT and either real or self-proclaimed IT skills or (b) entire groups of software professionals employed by business units outside of the control of the corporate IT department. Activities encompass a broad spectrum: office automation tool support; hardware and software purchase, deployment and administration; software development; etc.

A major difference between corporate and shadow IT is that the latter escapes *governance*. Governance can be broadly defined as *"the task of steering and coordinating actions of collective actors and managing the interdependencies of these actors"* [37]. When applied to information technology, the term governance lacks of a clear, shared definition. A systematic literature review of 12 definitions proposes the following compound definition: *"IT governance is the strategic alignment of IT with the business such that maximum business value is achieved through the development and maintenance of effective IT control and accountability, performance management and risk management"* [38], which when applied to information systems we summarize as follows.

definition **Information System Governance**
Ability to get an overview of the information system and drive its evolution in line with the corporations' objectives.

2.4.2. Shadow Applications as Evidence of Insufficient Agility

One consequence of the shadow IT phenomenon of particular interest with respect to information systems is that it introduces new applications to work around the shortcomings of the official ones. A recent experience report [39] contributes observations about this phenomenon in a 10 year long collaborative engineering project at The Boeing Company.

> *"Although many of the applications were bespoke efforts, designed to the requirements of users, virtually all major applications had an unofficial spreadsheet or database backing up the official application. These tools invariably played a critical but unofficial role in the day-to-day work, serving as more than just a workaround, whereas the official applications were used primarily for mandated record keeping and auditing."*

> *"Surprisingly, management often approved these unofficial applications but, at the same time, desired to eliminate these applications and use only the official applications."*

These observations match our own experience of 20 years of industrial information system development and integration in multinational corporations. The Boeing experience report contributes the following volumes.

> *"The complex IT infrastructure had at its core a suite of three integrated engineering applications, each customized for this program. A collection of about 30 secondary applications was tightly integrated with the core suite. These applications were widely used and mission-critical, but did not share the same management visibility as the core suite. The applications were either extensively*

customized or developed specifically for this program. In addition, approximately 500 applications were less well integrated and had specialized, domain-specific uses."

As a generalization, we can consider information systems of large corporations a web of numerous applications. At the center we find a fairly small set of stable and robust central applications. These are surrounded by a larger set of semi-official applications and a very large number of unofficial applications, for which we adopt the term *shadow application* proposed in [39].

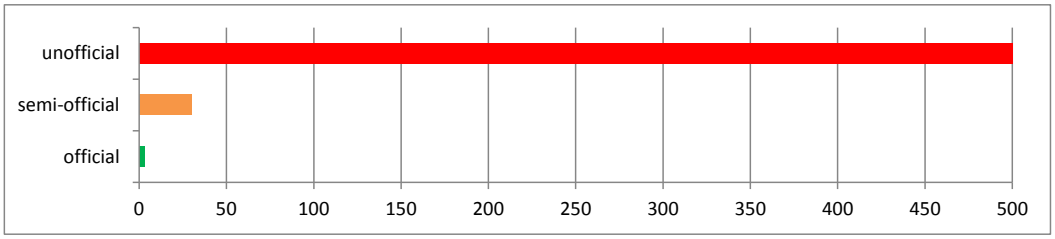


Figure 2-12. Relative numbers of official and shadow applications from [39]

Due to both our own background and our academic reference material, we will focus on use cases from manufacturing corporations. However, nothing indicates that shadow applications are restricted to manufacturing environments. Discussions with professionals from various domains (banking, telecommunications, healthcare and government) hint at a widespread, global phenomenon.

2.4.3. Motivating Example

In the remainder of this document, we will reuse fictional examples derived from the information disclosed by The Boeing Company in [39]. They describe an official application "Luxury", with the following limitations. Two shadow applications have been introduced to work around these shortcomings.

- **Delay analysis**
  "Luxury can report delays on process instances, but not the reasons for these delays which are managed by a shadow application."
- **Subtask tracking**
  "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 the Luxury application tracks *requests*, which is a common use case in engineering environments ("change request", "request for quote", etc.). The figure below shows a fictional official Luxury application, or more precisely a fictional form displaying a request object as managed by Luxury.

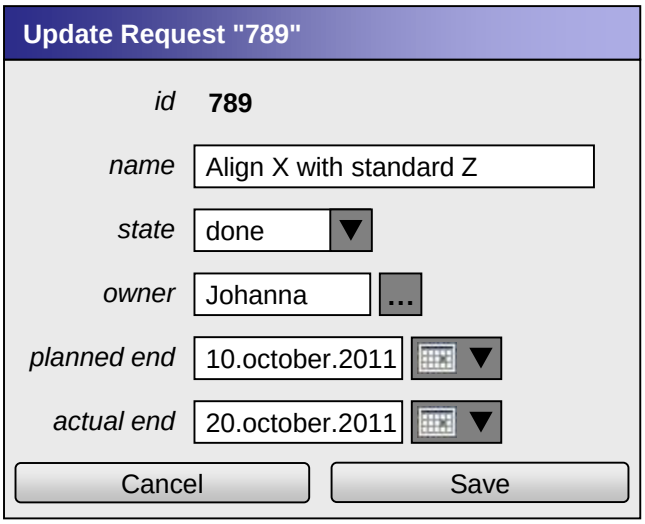


Figure 2-13. Example of fictional official Luxury application

The form shows that Luxury provides neither a way to record the reason for the delay nor a way to decompose a task into subtasks. The figure below shows two fictional shadow applications built by business units to overcome these limitations, managing delay analyses and subtasks respectively.

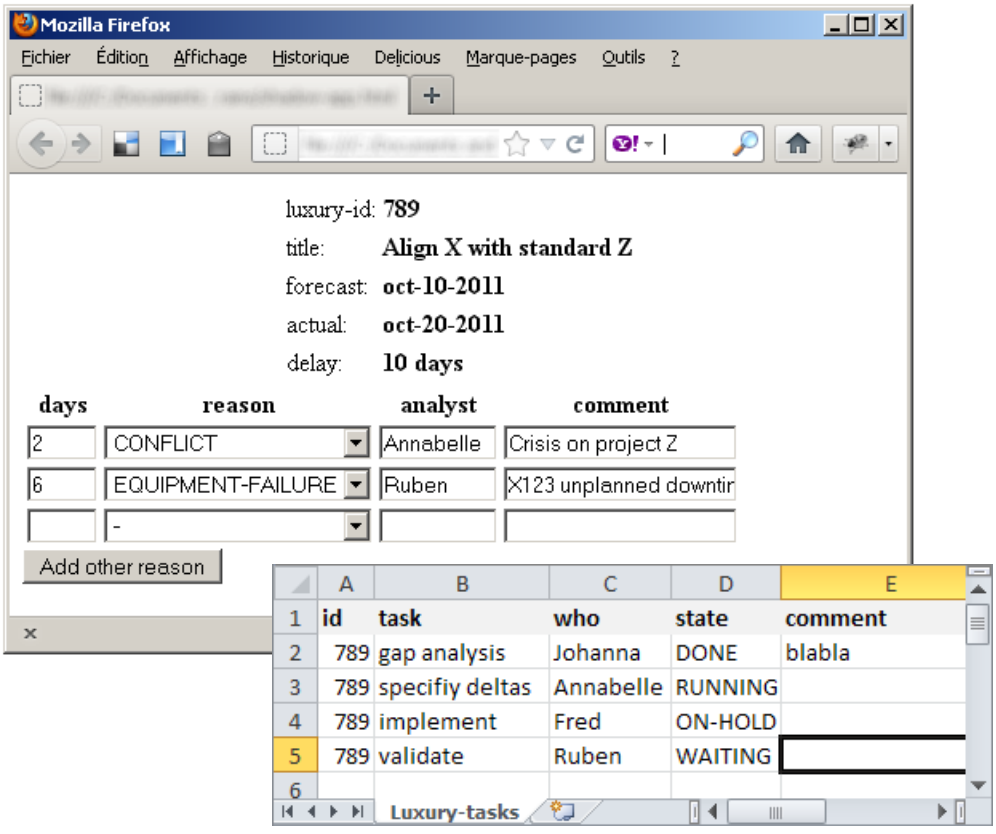


Figure 2-14. Examples of fictional shadow applications

The first author of [39] has confirmed in private communication that these fictional examples are "close enough" for the purpose of the demonstration.

Our example shadow applications are simple, the bottom application in Figure 2-14 being a simple spreadsheet. In our study we will frequently refer to spreadsheets designed and used simply for 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 [40, 41] and such spreadsheets qualify as shadow applications [39].

Spreadsheets can be considered to cover all three tiers of business applications, providing users with persistent tables, logic (formulas) and presentation tools. It is important to notice that despite their simplicity, such spreadsheet applications are vital for the daily operation of a corporation and that they store critical data and business logic outside of mainstream applications and controlled processes.

More Examples of Shadow Applications

Spreadsheets are arguably the most common form of shadow application, which we can consider as the low end of the spectrum. At the high end, shadow application architectures are limited only by the owner's imagination and resources. It is possible for resourceful business units to introduce full-blown business applications on their own budget. It is thus common, though not necessarily widely advertised, for big corporations to have *multiple* ERP or PLM systems[7]. More fashionably, actors utilize 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*" [42].

Shadow applications are both a problem themselves, as we will illustrate in section 2.4.5, and evidence of a bigger problem, i.e. the gap between the official information system and the daily needs of knowledge workers and groups.

2.4.4. Characterizing Shadow Applications

Shadow applications are characterized by their purpose. If application B exists to work around the limitations of application A, or if B's data and features belong in A according to its users, B can be considered a shadow application. This partial characterization illustrates the subjective nature of the phenomenon. With this characterization, "official" and "shadow" are relative concepts, and apply recursively at various levels of an organization. In other terms, multiple layers of shadow applications exist across the corporation, the final one being *personal* applications.

Shadow applications are also characterized by their ownership. If it's owned by the corporate IT department, it's an official application; otherwise it's a shadow application. The important aspect of this 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 to changing requirements without consulting other stakeholders or relying on the IT organizations' priorities. It also provides him with full control over the visibility of the data and access to features. It is worth noting that individual spreadsheets meet this definition.

[7] Besides the obvious cost drawback, this defeats the purpose of an ERP system which is to be the single referential for a certain domain, capturing the data and implementing the processes.

The diagram below illustrates both characteristics.

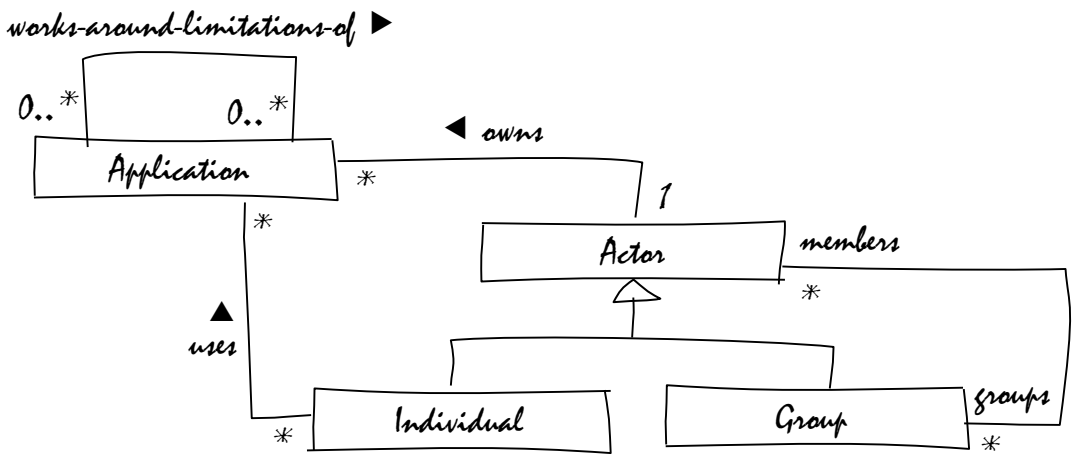


Figure 2-15. Application work-around and ownership relations

For the purpose of our study, we will retain the *objective* characteristic of shadow applications to define them, i.e. ownership.

definition **Shadow Application**
An application both functionally and technically owned by the actor using it.

An important characteristic of shadow applications is the visual integration of external ("official") data and shadow data. It is common for shadow applications to replicate a subset of the data from one or several official applications (often manually [43]), add some data and features, and provide forms and reports with a unified view of all data relevant for a given profile, which we define as *profile-specific*.

definition **Profile-Specific Application**
An application which unifies everything relevant for a given profile, and nothing superfluous.

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 [43], and having critical information and functionality scattered, unreachable and managed outside of standard IT processes is obviously not what comes to find when envisioning a well-structured and robust information system.

Shadow applications are usually considered a “necessary evil” [39, 43], filling the social-technical gap [31], i.e. the divide between what the corporate IT department knows it must support and what it can support technically. Corporations cannot work without them, but would really prefer to avoid the data duplication they imply as well as the burden they represent, as presented in the next section.

2.4.5. Benefits and Drawbacks of Shadow Applications

Shadow applications provide business-units with the agility they need (and thus the company needs) to survive and be successful. In this section, we describe the benefits of shadow

applications and the associated drawbacks. In order to do this, we evaluate shadow applications first from the “local” point of view of the actor owning them and second from a corporate, i.e. “global”, point of view. The local benefits of shadow applications initially outweigh the local drawbacks; otherwise business-units would not introduce them on their own budget.

The shadow application owners’ point of view

Probably the most important benefit from the owners’ point of view is the full *autonomy* to implement his specific needs, as soon as these arise, without other actors in a position to impede or slow down this implementation or to distort the needs by generalizing them. Also, unlike official applications, the shadow application owner decides about the visibility of his specific data and features to the rest of the world (*confidentiality*).

Shadow applications can *integrate*, either manually or automatically [43], data from other applications in order to provide a *uniform* profile-specific interface with all relevant information for a given profile or task. Integrating data in a single shadow application also ensures a certain *resilience* through independence of unreliable or slow related applications.

All or most resources, whether people, hardware, or software, are under the direct responsibility of the shadow application owner, who can thus decide about their allocation and be sure they are *dedicated* to his highest priority, which by construction cannot be guaranteed when resources are mutualized.

The main and possibly only drawback of shadow applications from a local point of view is that their owner must support all *costs* himself: purchase or development costs, integration and administration costs, hardware costs, and most importantly maintenance costs. Precisely the costs corporate IT is endorsing for the official applications.

Shadow applications can be of *lower quality* than official ones. They are managed by people who are not necessarily information system professionals and who often have other responsibilities. They can be developed by amateurs [44], with ad hoc or nonexistent processes, and the quality target is thus typically to be "good enough".

The corporate point of view

The main benefit of shadow applications from the corporate point of view is that they provide the necessary business *agility* for the survival and success of the corporation in a rapidly changing world. But besides this vital benefit, they present serious drawbacks.

Massive data *duplication* is the norm [43]. This implies *inconsistencies* and staleness, which are precisely what information systems try to avoid. Redundant features are common as well, and potentially inconsistent (for example different algorithms for computing the same business indicator).

Critical data and features are scattered, which not only generates the risk of losses and bad decisions but makes the overall information system *ungovernable*.

Various kinds of *waste* can be observed, like redundant development efforts. A *lack of awareness* at the global level is typical, preventing the propagation of best practices for which business applications are an ideal vehicle.

Most importantly, shadow applications end up *aggravating* the agility problem, due to the numerous replications of central data which amplify the effort of changing a central element, sometimes to the point of becoming prohibitive.

Summary

The table below summarizes the local and global benefits and drawbacks of shadow applications.

Owners' point of view	Corporations' point of view
☑ autonomy	☑ agility (i.e. survival)
☑ integration / uniformity	
☑ resilience	
☑ confidentiality	
☑ dedicated resources	
✕ cost	✕ duplications / inconsistencies
✕ variable quality	✕ ungovernable
	✕ waste
	✕ lack of awareness
	✕ **aggravate agility problem**

Table 1. Summary of benefits and drawbacks of shadow applications

Table 1 shows a high number of local benefits and few local drawbacks, the major one being the cost which is supported by the owner alone. The cost of business applications in general is often underestimated at the beginning of their lifecycle [45], which means that at their inception shadow applications present only benefits to their owner. Over time, the cost becomes a problem when the shadow application grows, hence the frequent requests for shadow application ownership to be transferred to the corporate IT department [39].

The summary also shows the high number of global drawbacks, and the single global benefit which justifies them.

2.4.6. Summary

Our definition of information system is the set of all applications running in a given corporation. This includes shadow applications, which is appropriate since these indeed manage vital aspects for the corporation as a whole.

Shadow applications are evidence of the lack of agility of information systems, but are not a solution due to the issues they introduce from the point of view of the corporation. However

they provide a set of requirements for a solution, both their benefits to be reproduced and their drawbacks to be avoided.

Before stating these requirements in chapter 3, it is interesting to try to understand the root causes for the lack of agility of official applications in the first place.

2.5. Factors Contributing to Insufficient Agility

Shadow applications initially emerge to work around the shortcomings of corporate applications [39, 36]. We thus need to understand the causes for these shortcomings.

2.5.1. The Requirements Paradox

In a naïve view of the world, providing a satisfactory business application would only require first correct requirements gathering and second the proper implementation of these requirements. While this may be achievable in simple situations, we think the difficulties described below make correct requirements gathering impossible in big corporations.

Distorted Requirements

Ideally, requirements analysis should be a highly social activity [46], bringing together all stakeholders, letting them express their requirements and expectations and converging on a common solution. Even for internally developed applications this is unrealistic due to the number of stakeholders, and for commercial software this is impossible since the most important stakeholders are usually not known in advance (future customers). Hence the common practice of expecting a subset of *user representatives* to faithfully and completely express requirements of other stakeholders, and trusting a few central roles to consolidate this into a single, consistent, satisfactory whole.

Regardless of the competence and motivation of both the user representatives[8] and the consolidators, in complex business environments the result is invariably that requirements are incomplete and distorted.

Conflicting Requirements

Large organizations are not consistent and orderly systems. Some level of conflict is useful for identifying and assessing options and avoiding counter-productive conformity ("groupthink") [47]. Referring both to groups and individuals, [48] describes working relationships as "*multivalent with and mix elements of cooperation, conflict, conviviality, competition, collaboration, control, coercion, coordination and combat (the c-words)*". When there are hidden or conflicting goals, people will resist articulating these [31]. Requirements from different stakeholders are thus incomplete and divergent [49] or downright conflicting [50].

[8] [175] indicates that user representatives express second-hand or third-hand knowledge at best, and are typically chosen among junior workers, the most experienced people being too busy and in short demand.

With big numbers and diversity of participants, requirements engineering means lengthy discussions, leading to delays and more or less acceptable compromises. In other words, actors must invest a lot of energy to get an unsatisfactory solution, late.

Continuously Evolving Requirements

As an aggravating factor, corporations are not static. They must adapt to changes in their environment like new markets, competitors, partners, technologies or regulations, with potential impact on their information system.

Change is not necessarily due to the environment. Over time, professionals learn better ways of doing their job and want to adjust their way of working accordingly.

Although the impact of the aforementioned c-words is most 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.

Summary

Requirements are distorted, conflicting and continuously changing. Under such circumstances, it is a challenge to converge on a consistent set of requirements and deliver a working application at all. Widespread dissatisfaction with the result is almost guaranteed by construction, which we call the *requirements paradox*.

definition

Requirements Paradox
The more stakeholders are involved,
the less the result is likely to satisfy anyone at all.

We consider this phenomenon paradoxical because the purpose of involving stakeholders is to satisfy their needs. But the principle does not scale in complex organizations.

2.5.2. The Knowledge/Influence Paradox

A problem with enterprise software is that the most important decisions are taken by the people farthest away from the real-life problems the application needs to solve.

Although end-users have the best knowledge of their information processing needs, they have little influence on the software applications they work with. By contrast the developers have a tremendous influence on the application, but they usually have insufficient knowledge on the details about application usage in particular contexts. We refer to this phenomenon as the Knowledge/Influence Paradox. The following figure shows a typical set of roles, with their respective influence on the application and their knowledge of the actual problem.

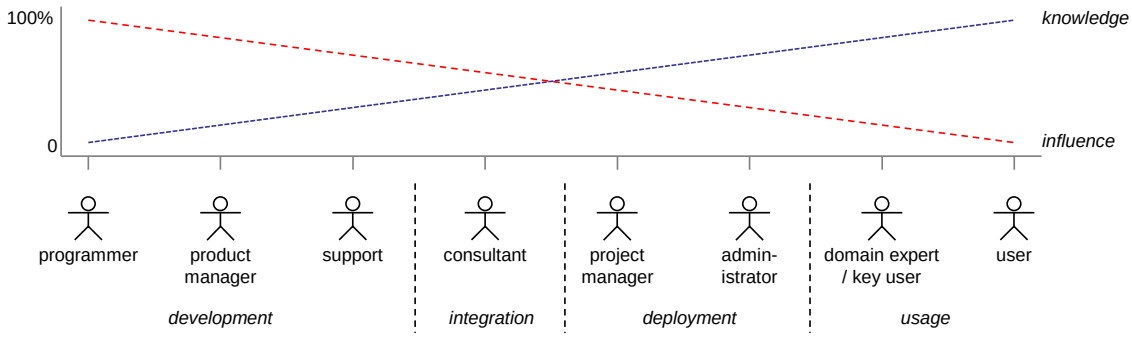


Figure 2-16. Typical roles with relative knowledge and influence

Besides illustrating the Knowledge/Influence Paradox, the figure above illustrates the idea of *distance* between the user and the developer. This distance can be 0, in the case of an end-user developing his own application. It is small in the case of shadow applications. And it can be much bigger than what is illustrated in Figure 2-16 in the case of commercial enterprise applications. The vertical dotted lines represent barriers, which range from non-existent in the case of shadow applications to multiple corporation boundaries in an outsourcing scenario with different companies developing, hosting, supporting, integrating and using enterprise applications.

definition

Knowledge/Influence Paradox
The closer the actors are to the problem being solved, the less influence they have on the application.

2.5.3. Secondary Factors

Besides the requirements paradox and the knowledge/influence paradox, there are a number of secondary factors contributing to poor agility of official applications and shadow application emergence.

The aforementioned widespread practice of *IT cost reduction* lowers both the reactivity and quality of IT support, inciting business units to help themselves [29].

Besides conflicting requirements, 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).

We could even add a new c-word, *creativity*, to the list. Application development is a highly creative activity, and problem-solving one considered a hobby by many, which can lead individuals to prefer the shadow application option for no other reason.

Technical obsolescence, a consequence of either respectable age or an unfortunate choice of foundation technologies, can make it difficult to find the right skillset to implement changes in existing applications.

Fear of change, either from the application owner or other users, can lead to immobilism. This natural phenomenon is aggravated by the fact that most organizations do not reward risk but do punish mistakes. Both actor A requesting a change and actor B approving it take the risk of

making a "public" mistake. Actor A changing or adding a shadow application is much less risky, failure would be limited to his own perimeter and probably unknown outside of it.

2.5.4. Inter-relationships of Factors

The flowchart on the right illustrates a number of typical obstacles for a new requirement on the path to implementation in an official application, as well as example reasons for choosing the alternative path leading to the implementation of the requirement in a shadow application.

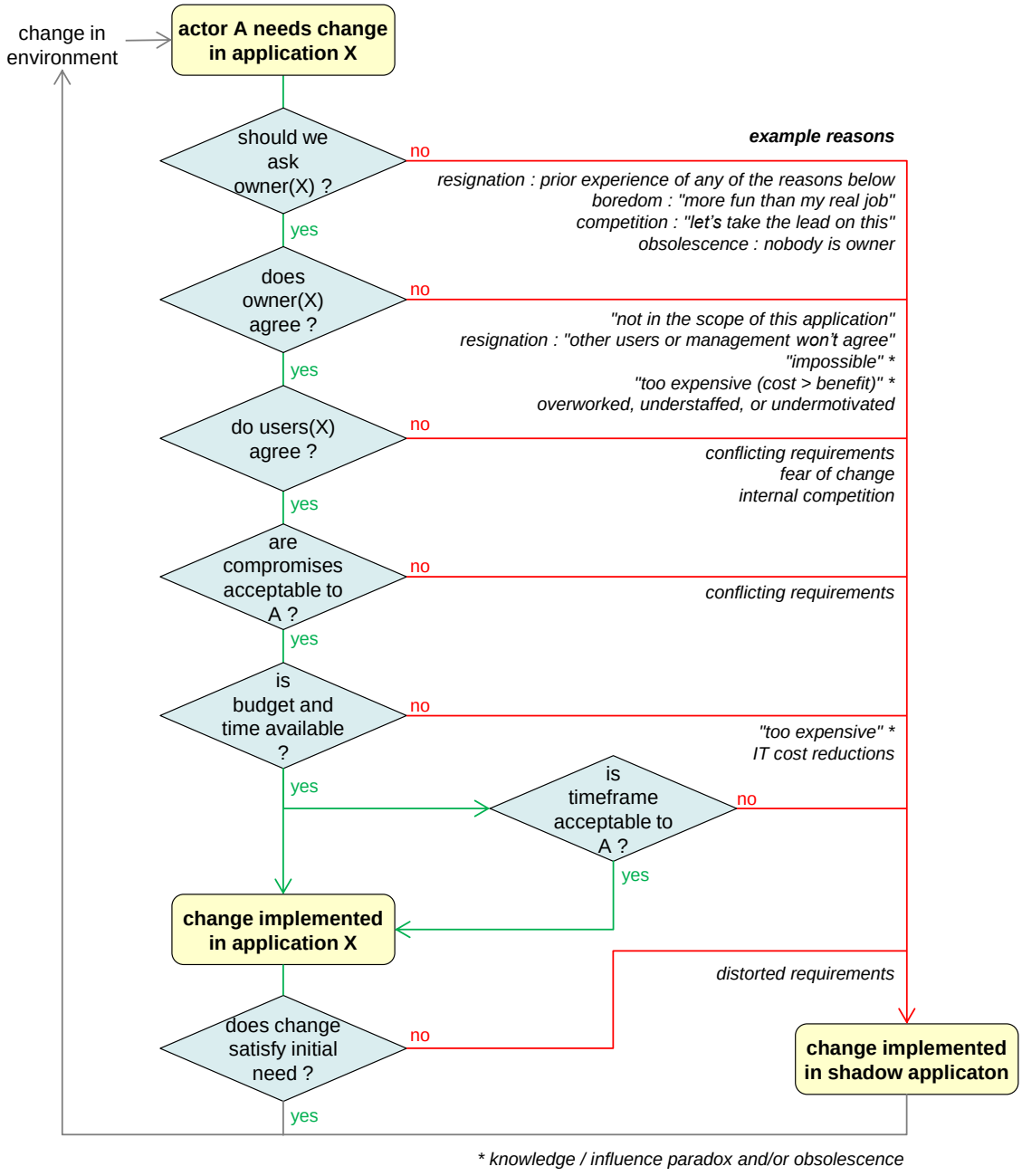


Figure 2-17. Paths to implementation of a new requirement in official or shadow application

The path for a given requirement to being satisfactorily implemented in the relevant official application counts seven hurdles. Even when adopting the optimistic probability of success of 80% at each step, the probability of a given requirement being satisfactorily implemented in

the relevant official application is only of 0.8^7=0.21, i.e. a 0.79 probability of being implemented in a shadow application. A neutral scenario with a 50% chance of success or failure at each step brings the probability down to 0.5^7=0.01, and thus an almost certainty of the requirement being implemented in a shadow application. These example figures are consistent with the volumes mentioned in [39] and represented in Figure 2-12.

Several interesting observations can be made with respect to the chart in Figure 2-17.

- After a first iteration through the shadow application path for a given actor, it becomes the path of least resistance for subsequent, potentially unrelated requirements.
- After the first step, branching to the shadow application path does not necessarily interrupt the official application path, potentially resulting in significant waste.
- Even when the official application path is successful, cycle times measured in years are not uncommon.

2.5.5. Summary

The previous sections have described a number of causes for lack of agility of official applications. Over time this leads to an overall state of disappointment and resignation of actors with their unsatisfactory information system [30, 51] and thus to shadow application emergence.

Though our analysis of causes is far from exhaustive, some patterns and commonalities emerge, which will be reused to formulate our research hypotheses in chapter 3.

2.6. Quantifying the Lack of Agility

This section aims at providing some measures and estimates in terms of shadow application numbers and costs.

2.6.1. Measuring the Number of Shadow Applications

The very nature of shadow systems implies that there is no central inventory where they can be easily counted. Measuring their numbers in real-life corporations is extremely difficult, due to a lack of incentive for cooperation both at the level of the corporation and at the level of groups or individuals.

Information systems are gaining strategic importance, embodying an organization's know-how and culture [12]. Big organizations - public or private - are therefore reluctant to publicly admit shortcomings in their information system, or to dedicate resources to an audit revealing information system deficiencies. The aforementioned report from The Boeing Company [39] is a notable exception.

In a certain sense, measuring shadow application development and usage is measuring a degree of taboo [52]. We have not found an experiment likely to yield accurate quantitative data about shadow applications in existing corporations (see appendix B), and our study will thus use the figures from the Boeing Company [39] and the estimates presented in the next sections.

2.6.2. Estimating the Number of Shadow Applications.

We can use publicly available data to estimate the number of shadow applications in other existing organizations.

The Boeing Company

In [39], the Boeing Company reports an organization with 3 official applications, 30 semi-official applications and 500 unofficial applications. A generalization of this observation could be to consider a 10x factor per "layer".

International Business Machines Corporation (IBM)

A public interview of the Chief Information Officer (CIO) of the IBM Corporation describes an internal application consolidation effort [53] of unspecified length and cost. The result was to reduce the number of applications supported by the corporate IT department from 16 000 to 4 500 today, with a final objective of 2 250 applications. The person clearly states that only a small subset of the business units' requirements will be implemented.

> *"The IT department may get requests for '100 new requirements' for the global application, which they will negotiate down 'to the 20 requirements that you are actually going to implement'".*

This roughly says that discarding 80% of expressed requirements is considered acceptable.

The interview also clearly mentions their acceptance of shadow applications and the choice of the IBM corporate IT department to provide dedicated resources, for which "*100 000 users have registered*".

2.7. Conclusion

Corporations must be agile to survive, and agility is thus a prime requirement for their information systems. The state of fragmentation we have described is proof of a serious lack of agility of today's information systems. Our opinion is that with present software architectures, no matter how carefully business applications are crafted, over time they will spawn shadow applications whenever resourceful actors have urgent unsatisfied needs, solving a local problem but aggravating the overall situation.

Our hypothesis is that a fundamentally different information system paradigm is possible, and that shadow applications provide insight into what organizations need. In the next chapter we express the benefits and drawbacks of shadow applications and the causes for their emergence as high-level requirements for an agile information system.

3. Requirements for an Agile Information System

This section presents our research hypotheses. Building upon the observation of the present state of information systems described in the previous chapter, we propose a set of high-level requirements for an *agile information system*.

Starting from a generalization of the benefits of shadow applications, section 3.1 describes **application requirements**, i.e. requirements which must be met by a single application. Section 3.2 presents **information system requirements**, i.e. requirements which must be met by the collection of all applications. A third section describes additional **derived requirements** related to composition, traceability, collaboration and governance.

Requirements will be numbered **R*x***, from R0 to R15.

3.1. Application Requirements

When facing a new business requirement, actors must be able to quickly reflect this change in the business applications they use. The first potential hurdle is a lack of influence.

R0	**Influence**

Whenever facing a shortcoming in an application, actors must be able to either adapt the application themselves or introduce a new application. We will refer to such changes as *adaptations*. In a data-centric vision, this encompasses adding attributes to existing business concepts, adding new concepts, changing multiplicities, introducing or removing states in a state machine, or more generally reducing or increasing any level of granularity.

EXAMPLE
In our Luxury example, an engineering organization adds a new concept "SubTask" with a reference to the existing concept "Request", effectively adapting the initial Luxury application.

EXAMPLE
Besides this organization-wide adaptation, individual employee X chooses to extend the existing concept "Request" with a new attribute "difficulty" in order to organize his work.

Allowing any actor to make changes to any application implies a mechanism which isolates actors from one another, which dictates requirement R1.

R1 Isolation

Actors must be able to introduce adaptations without impacting other actors. The rationale for this requirement is that seeking agreement with other stakeholders on the specification is not only time-consuming but likely to fail (see section 2.5). This does not imply that adaptations are private and that other actors cannot see them (which is a separate concern covered by R4) but rather that actor A's adaptations are not applied to actor B by default.

EXAMPLE
In the examples in section R0, the sales organization does not see concept "SubTask", and likewise, no individual besides X sees attribute "difficulty".

These first two requirements introduce *application variability*. Depending on the actor, a different set of adaptations will apply, which is a first step towards *profile-specific* applications.

Other common hurdles between the expression of a requirement and its implementation are *hardware resource limitations*, *skills* and *confidentiality*, which dictate the next 3 requirements.

R2 Hardware Independence

R0 and R1 provide any actor with the freedom to adapt applications owned by another actor. However it would not be realistic to expect application owners to provide sufficient resources to support the burden of all adaptations by other actors. As a consequence, adaptation owners must be able to host their adaptations on their own hardware resources, i.e. resources they can purchase and administrate themselves. Failing to meet this requirement would put the application owner in a position to impede the deployment of adaptations for resource limitation reasons, or even political reasons. R2 is effectively a prerequisite for R0.

EXAMPLE
If the engineering department wants to attach big specification documents and CAD files to Tasks or SubTasks, it should be possible to host these documents on the engineering departments' own hardware resources.

R3 Ease of Modification

Even without having software development skills or having the budget for hiring or subcontracting someone who does, any individual must have the possibility to implement adaptations himself, following the *gentle slope principle*, which states that in order to modify an application through its user interface, end users should have to *"only increase their knowledge by an amount in proportion to the complexity of the modification"* [54, 55].

Simple adaptations must not require programming. More complex tailoring tasks should be possible with user-oriented programming languages and building blocks [56]. Professional software development must be possible as well.

EXAMPLE
In order to add the new attribute “difficulty” to existing concept “Task”, user X could simply click next to the last column, type “difficulty” as a column header providing the attribute name, and start filling values which can help determining the data type of the attribute. Spreadsheet-like formulas provide a way to express simple processing, and wizards can assist more complex operations like adding a new association between two concepts.

R0 (influence), R1 (isolation) and R3 combined reduce both the requirements paradox and the knowledge/influence paradox: everybody can easily change what needs to be changed to do his job without impacting others.

R4 Confidentiality

An actor who introduces information system elements must be able to control their visibility. A first reason for this is to implement business-related confidentiality rules, i.e. only exposing sensitive data on a need-to-know basis. A second reason is to prevent time-consuming and counterproductive post-implementation arguments with other stakeholders who might disagree about political aspects like ownership, or about functional aspects like the business process.

EXAMPLE
In our case of a Request management system, a first example is a sales department who could decide to add an attribute “expected sales” in order to prioritize their work. They prefer to hide such very rough projections from other actors.

EXAMPLE
As a second example, the corporate quality process dictates that for all requests which last longer than 5 days, a “retrospective” meeting is required. The engineering team considers it is a waste of time to do this systematically, and decides to add a boolean attribute “needs retrospective” to mark for which requests they can bypass the process. Such an attribute clearly contradicts the official process, and in order to prevent lengthy explanations they decide to hide the attribute from the rest of the corporation.

We consider that if anybody (R3) can quickly change whatever he wants (R0) without necessarily consulting (R1) or even informing (R4) other actors, with his own hardware resources (R2), an application supports *conflicting requirements*. We have shown that due to organizational complexity, requirements are divergent by construction, and supporting them greatly reduces the requirements paradox.

3.2. Information System Requirements

In an information system composed of applications meeting requirements R0-R4, a great number of actors can introduce changes; it is worth remembering the Boeing figures of 530 shadow applications for 3 official applications. This section expresses desirable properties of the overall resulting information system: *consistency*, *resilience* and *uniformity*.

R5 Consistency

Duplication must be avoided, both in terms of data and of features, resulting in a system consistent by construction. This may seem straightforward, but observations of real-life information systems [43, 39] make it worth stating.

EXAMPLE

In the first example for requirement R4, the values for the new "expected sales" attribute (added by the sales department) must be stored without even partial replication of official Request records.

R6 Resilience

In a scenario where many actors can provide application elements (R0) and host these on their own hardware resources (R2), variable reliability is a certainty. Network availability and performance is often uncertain as well, especially when considering small remote sites. It is important that the information system is *resilient* when parts are down, i.e. unreliable or temporarily missing parts must not preclude a user interacting with the available parts.

EXAMPLE

A central application hosted in Europe (on server S_E) has been extended with a set of attributes by a business unit X, which has hosted these extensions on a server in their biggest location, i.e. in Asia (server S_A). When employees from European or American sites of business unit X interact with this application, a slow or broken link with the Asian continent should not impede access to the elements on server S_E.

R7 Uniformity

Actors should not have to deal with many different user interfaces. Ideally, they should have a single point of access which seamlessly integrates all data and features relevant to their job, and no more. It is thus important to blend together applications and the relevant adaptations into a uniform user interface (the aforementioned profile-specific application), to avoid juggling with numerous different user interfaces.

EXAMPLE

When a person from the quality department looks at a Request, he sees both the official data (title, dates …) and the data which is specific to the quality department (delay analyses), as illustrated in the top form in Figure 2-14.

EXAMPLE

The bottom example of the same figure provides a counter-example: people from the planning department need to look both at the official application (Figure 2-13) *and* their extension spreadsheet to get the overall picture.

3.3. Derived Requirements

3.3.1. Composition Requirements

Whether looking at our previous description of information systems (see Figure 2-11) or at requirements R0-R7, we consider a business application as a *composition* of elements. This composition typically happens at development-time and is thus manual and static. We propose the next two requirements as desirable characteristics of the composition mechanism.

R8 Profile-driven Composition

Today, applications typically use the users' profile (in particular organization memberships and roles) to filter the elements which are available to him. Beyond removal of forbidden elements, user profiles should *drive* the composition mechanism, automatically including relevant adaptations.

EXAMPLE
Actor X is working in the planning department, and as a result the "subtask" adaptation is part of his daily work. He has recently been named "quality champion" for his department, and as such he is now interested in adaptations regarding quality aspects. His applications should automatically reflect both the planning and the quality adaptations, without the need for human contribution.

R9 Change Propagation

Application elements can be adapted (R0). When the original element changes, this change should be reflected in all its adaptations without human contribution, resulting in the same situation as if the attribute had been present from the start. This means that applications should either be dynamic or automatically rebuilt in the case of an "upstream" change. Traceability is a prerequisite (R10-R11).

EXAMPLE
The owner of official application Luxury introduces a new attribute "business value". Even though the new attribute has been added after the extensions, the applications from the quality and planning department must automatically reflect this new attribute, without human contribution.

3.3.2. Traceability Requirements

Evolution of complex information systems is not trivial, especially when envisioning distributed ownership of elements. Pre-implementation impact analysis is an important aspect of change management. The following two requirements propose bi-directional traceability to enable such impact analyses. We adapt the terms forward and backward traceability from requirements engineering [57] to our proposal.

R10 Forward Traceability

When considering a necessary evolution or removal of an element, understanding its usage is mandatory to measure the impact of the change. We call forward traceability the possibility to

determine where a given element is used. Commonly advocated enterprise architecture principles like layering [58], loose coupling [59] and separation of concerns don't necessarily provide the owner of an element with this visibility.

EXAMPLE
In the example illustrating requirement R9, while considering the introduction of the "business value" attribute, the owner of Luxury can inspect adaptions before the change, and possibly contact the owner of a similar extension to discuss potential conflicts together beforehand.

R11 Backward Traceability

When interacting with a profile-specific application, a user manipulates a composition of. He should be able to know the origin of all these elements, in order to get explanations, suggest evolutions, or determine trustworthiness with respect to his particular concern.

EXAMPLE
A user from the quality department notices that the “delay” calculation is incorrect (Figure 2-14) because it does not take into account weekends. He should be able to determine whether to notify the problem resides in the official application or in the quality-specific adaptation.

3.3.3. Collaboration Requirements

When an actor faces a new requirement, he can either request a change in another application (R11) or decide to implement the change himself (R0). However, instead of deciding to introduce a new element of his own, he could reuse an existing adaption. In order to minimize duplicate efforts, our last set of requirements deals with two aspects of collaboration: sharing and awareness. These are closely related to R5 (consistency).

R12 Sharing

R4 ensures that actors have the possibility to hide their elements from other actors if they wish. However, when the owner considers an element mature enough and potentially interesting for other actors, he must be able to *share* this element, making it available to a wider group of actors in order to minimize duplicate efforts and to propagate best practices.

EXAMPLE
Employee X has extended concept "Request" with an attribute "difficulty" in order to organize his daily work by descending difficulty. When colleagues from his team wish to adopt this approach, X shares this adaptation with them. After a few weeks of positive impact on the teams’ productivity, employee X decides to share the “difficulty” adaptation with a broader group of employees.

R13 Awareness

The corollary of R12 is that actors need to be *aware* of all existing elements available to them, through both search (“pull”) and notification (“push”) mechanisms.

EXAMPLE
Employee Y wants to better manage his time, and wonders how he could better organize his tasks. He should be able to easily find employee X's "difficulty" extension, so he can adopt it if it fits his way of working, instead of introducing a similar extension.

EXAMPLE
More and more actors adopt the "difficulty" extension. An employee Z with a similar profile who hasn't yet thought of prioritizing his tasks in this way could be proactively notified of this popular feature.

R14 Relevance

As soon as an element is shared by its owner, it is available to other actors, and potentially relevant for their activity before they identify the need. However, in a big organization where all actors can contribute and share, the potential number of available elements is huge and information overload must be avoided, especially in the case of notification mentioned in R13 (awareness). It is thus important to provide assistance to actors to determine which elements are most *relevant* for them, not only enabling but accelerating the propagation of best practices.

EXAMPLE
If aforementioned actor Z (R13) gets notified more than once about available extensions which are not relevant for his job, he will start ignoring subsequent notifications. As a corollary, if most suggested elements so far were relevant, he will pay close attention to new notifications.

3.3.4. Governance Requirements

The previous requirements have depicted a situation where many elements are introduced and shared by many actors. Our final requirement deals with governance, according to our previous definition.

R15 Governability

Corporate management must be able to get a clear overview of the information system as a whole and of the dependencies between its components, in order to drive its evolution. Several previous requirements can be considered as prerequisites for governability, especially R5 (consistency), R10-R11 (traceability) and R13 (awareness).

EXAMPLE
The popularity of the "difficulty" extension should be brought to the attention of the corporate IT department, in order to assess whether it could be useful beyond its present adopters and proposed for wider use, and possibly retire existing alternatives which nobody uses.

3.4. Conclusion

In this chapter we have expressed the generalization of the benefits of shadow applications in the form of requirements for an agile information system, and have complemented these with further requirements in order to avoid their drawbacks. The resulting set of requirements for an agile information system is summarized below.

Category	Requirement
Application	**R0**: Influence
	R1: Isolation
	R2: Hardware Independence
	R3: Ease of Modification
	R4: Confidentiality
Information System	**R5**: Consistency
	R6: Resilience
	R7: Uniformity
Composition	**R8**: Profile-driven Composition
	R9: Change Propagation
Traceability	**R10**: Forward Traceability
	R11: Backward Traceability
Collaboration	**R12**: Sharing
	R13: Awareness
	R14: Relevance
Governance	**R15**: Governability

Table 2. Summary of requirements for an agile information system

In the next chapter we evaluate the two dominant information system architecture paradigms versus these requirements, and describe related research work which provides elements for a solution.

4. State Of The Art

In this section, we evaluate the two dominant information system architecture paradigms with respect to our requirements for business agility: **application-centric** and **service-oriented**. We discuss further research work related to **requirements complexity**, **business agility**, and coping with **information system fragmentation**

4.1. Information System Architecture Paradigms

The most common architecture paradigm is centered on *applications*. In this vision, the information system is a collection of more or less integrated applications, which users interact with through various application user interfaces. We call this paradigm *application-centric*. The diagram below shows that the primary decomposition is *vertical*, with a secondary decomposition along the tiers presented in Figure 2-5.

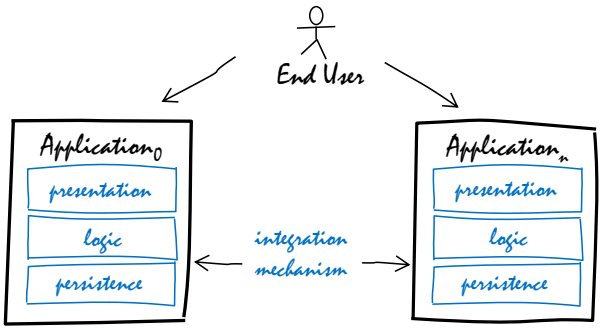


Figure 4-1. Application-centric architecture overview

An alternative, more recent architecture paradigm revolves around *services*. Services are components which expose business or technical capabilities to be invoked by other components. Services can be complemented by *widgets*, i.e. snippets of user interface providing a way to interact with a service. We call this paradigm *service-oriented*. The diagram below shows that the primary decomposition is *horizontal*, with a secondary decomposition along domain boundaries.

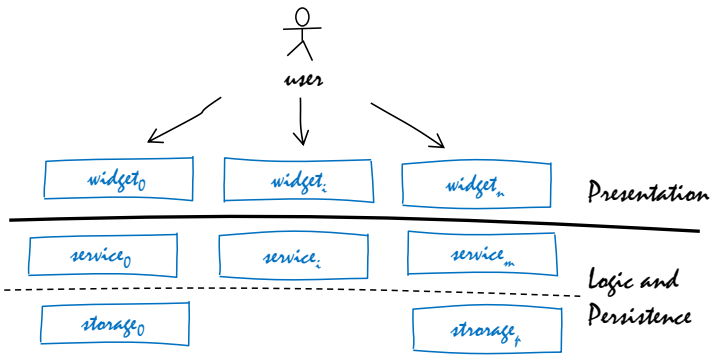


Figure 4-2. Service-oriented architecture overview

These two paradigms are not mutually exclusive. Applications have been gradually adopting more open middle tiers, effectively exposing the logic as services in order to facilitate their integration in service-oriented architectures. Services on the other hand are frequently used as a foundation to build applications.

Each information system is unique, with its own mix of closed monolithic applications, more open service-enabled applications, services and (less frequently) widgets. It is thus impossible to evaluate all possible combinations. In order to evaluate the two paradigms, the next two sections describe and evaluate the two extreme scenarios: purely application-centric and purely service-oriented.

4.2. Application-Centric Architectures

Application-centric architectures are the dominant situation in corporations today. The vast array of methodologies and technologies for application development and integration precludes an exhaustive state-of-the-art. In this section we first provide a more detailed description of our assumptions about the typical characteristics of an application-centric information system, and then evaluate it versus our requirements for business agility.

4.2.1. Description

An application provides a complete solution for a given problem domain. It covers the full stack from persistence to presentation and is thus often referred to as a vertical solution, as illustrated in Figure 4-1.

Big corporations have multiple applications, configured for their specific needs [60]. In theory a single very powerful application could fulfill all needs. In practice however, even at the corporate level more than one enterprise application is usually present, for either historical reasons (mergers or acquisitions), functional reasons (no single application can meet all requirements), legal reasons (different regulations depending on country or target business), technical reasons (obsolete platforms with business-critical modules) or even strategic reasons ("best-of-breed" approach [61]).

Within a corporation, the same real-world business concept (a product for example) typically exists in various applications, i.e. it has multiple and different *representations*. In an ideal information landscape, only one clearly identified application owns the *master* representation for each instance of a concept, and all other applications have *replicas*.

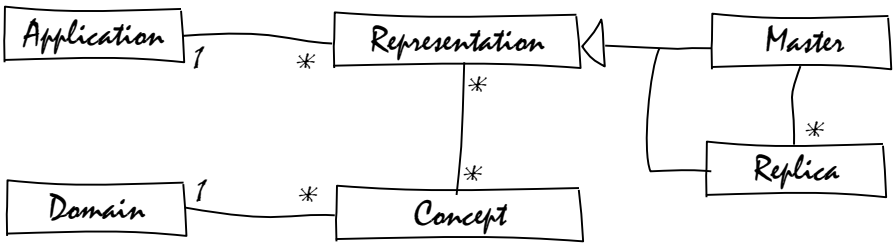


Figure 4-3. A real-life Concept and multiple Representations in various Applications

It is possible to have several applications owning master data for one concept. The simple case is a horizontal partition, when a given instance belongs to only one application according to its

nature. For example, in a company producing both hardware and software products, it is not uncommon to have separate master applications for each. The more complex case is vertical partitioning, where the same instance exists in multiple applications with different attributes, each master of its own set.

Synchronizing replicas with master data can be manual [43] or via a form of *integration,* which we will discuss in section 4.6. It implies extracting the master data and inject it into other applications which can then operate on local data. An alternative to this data replication would be to design applications in a way permitting to operate on remote data. This was the main objective of the distributed objects paradigm [62], which has slowly evolved into the service-centric scenario we will discuss in the next section. Modern business applications have a well-defined and documented service interface allowing to extract or inject data, and to invoke processing. Most applications have a "one-way" integration philosophy, where they consider themselves as the focal point for a given problem domain and operate on local data.

The core set of integrated official applications is typically owned by the corporate IT department, and only represents the tip of the iceberg. As described in section 2.4.2, in order to meet their business goals, business units complement this set of official applications with their own semi-official applications. These can exploit the core integration mechanisms to synchronize reference data with official applications. Recursively, groups and individuals in turn introduce further levels of unofficial applications to do their daily job; in order to evaluate the application-centric scenario versus our set of requirements, we thus adopt the following rough definition.

definition **Application-centric Information System**
Information System where a requirement is implemented within a given application, official or shadow.

The next section evaluates application-centric information systems as a whole. We will provide separate ranks for official applications and for their unavoidable shadow counterparts, and will derive a compound rank for an application-centric architecture, as illustrated by the table below.

requirement	official applications	shadow applications	application-centric architecture
Rx <label>	$rank_{official}$	$rank_{shadow}$	*compound rank*

Table 3. Explanation of application-centric ranking columns

4.2.2. Evaluation

When an actor A faces a new business requirement R which needs to be reflected in a given application, two situations can occur: either the application in question is owned by an external group, or it is owned by actor A himself. In the first case, actor A must request a change from the application owner (see Figure 2-17). The result, if any, is rarely satisfactory, often leading to the implementation of requirement R in an existing shadow application or even the emergence of a new one. In the second case, i.e. actor A owns the application, he can directly implement the requirement R in his shadow application.

Thanks to the existence of shadow applications, actors have full freedom to implement any change at will, without disrupting other actors. This satisfies both requirements **R0** (influence) and **R1** (isolation). Shadow applications typically run on business-unit owned hardware, satisfying **R2** (hardware independence) as well.

Numerous tools exist to allow individuals with varying degrees of software development skills to implement business applications on their own. The most popular are spreadsheets, which allow to represent structured data and express simple processing with minimum skills, while providing more software-literate individuals with a rich set of processing and even communication functions. Spreadsheets are thus an excellent illustration of the gentle slope principle [54, 55]. Tools like Microsoft/Access target the development of small-scale local database applications through a form-based development environment. Online tools like Intuit/QuickBase or Tibco/FormVine provide similar functionality and ease-of-. Application-centric environments thus satisfy requirement **R3** (ease-of-modification), again thanks to shadow applications.

Shadow applications satisfy **R4** (confidentiality) by definition. Owners of shadow applications can clearly decide who they grant access to. It should be noted here that corporate IT departments can almost always access everything: they typically have administrator access on all machines, both on servers and on all of the organization's personal computers.

The table below summarizes the rankings of application-centric architectures versus R0-R4.

requirement		official applications	shadow applications	application-centric architecture
R0	influence	✗	☑	☑
R1	isolation	✗	☑	☑
R2	hardware independence	✗	☑	☑
R3	ease-of-modification	✗	☑	☑
R4	confidentiality	✗	☑	☑

Table 4. Ranking of application-centric architectures versus application-level requirements

Considering that R0-R4 are generalizations of shadow application characteristics, this summary is no surprise. However, it clearly illustrates how shadow applications compensate for the lack of agility of official applications.

When considering an application-centric information system as a whole, inconsistency is a main weakness. Due to numerous replication paths, with successive selections and projections (in the relational algebra [63] sense of the terms) and transformations, and multiple manual entry points for identical business concepts, consistency is impossible by construction. This is not obvious when looking at a single application, which is a consistent entity with well-implemented integrity constraints. When trying to integrate several systems, discrepancies are usually discovered [64]. The whole information system depicts a scary picture.

This situation is made tolerable by the human factor: people can deal with inconsistency. But we consider that consistency has been sacrificed to achieve necessary agility, and application-centric architectures fail to meet requirement **R5** (consistency).

Applications, whether official or shadow are self-sufficient entities. As such, they are usually designed to function independently from the availability of other applications. This loose or non-existent coupling results in an overall resilient system, where the failure of one particular application has little or no impact on the remaining applications. Also, in multi-national environments with networks of variable speed and reliability, the replication mechanisms which applications naturally encourage have the beneficial side-effect of isolating single-application interactions from network problems. We can thus consider that application-centric architectures satisfy requirement **R6** (resilience).

When a users' concerns span multiple applications, he usually interacts with all these applications in turn, through distinct interfaces. This clearly fails to meet requirement **R7** (uniformity). However, shadow applications often pull together the relevant data elements for a given task. They present both local master data and replicated data in a convenient unified interface, optimized for a given actor or task, which we can consider satisfies R7 in read mode. Data updates are typically local. Although it is sometimes possible to implement bi-directional synchronization by invoking services when these exist, the update of master data often requires manual changes in the source application. We will thus consider that requirement R7 is not satisfied in write mode, as reflected by the double rank below for read and write (r / w).

requirement		official applications	shadow applications	application-centric architecture
R5	consistency	☑	☑	✗
R6	resilience	☑	☑	☑
R7	uniformity (read/write)	☑	☑ / ✗	☑ / ✗

Table 5. Ranking of application-centric architectures versus information system-level requirements

Intrinsically, business applications are standalone entities, composed manually at development-time. If an actors' profile involves multiple roles spanning several applications and adaptations, these do not get automatically composed without human contribution. Requirement **R8** (profile-driven composition) is thus not satisfied.

Applications, both shadow and official, have local replicas of data they do not own. If the master schema evolves, the synchronization mechanism can break but even if it doesn't, the evolution will typically not be dynamically propagated to other applications. We thus consider that requirement **R9** (change propagation) is not satisfied.

Applications are primarily designed for human users. Users are authenticated, and application owners are thus usually aware of who uses his application. This is not necessarily true for outgoing integration mechanisms. If this mechanism is a read-only database access or a service interface invocation, it is possible to keep track of which other applications depend on a given application through authentication. In the case of file extracts published in an area with wide access, or messages published on communication middleware, this may be difficult, or over

time too impractical and expensive, or even impossible. By adhering to the principle of decoupling [59], application-centric systems do not satisfy requirement **R10** (forward traceability).

Because applications mostly or only manipulate local data, little importance is given to the origin of this data. Users of application B may be aware that some data is replicated from application A, but such traceability is typically not a primary concern and no online documentation or mechanism is provided. This fails to satisfy requirement **R11** (backward traceability).

requirement		application-centric architecture
R8	profile-driven composition	✗
R9	change propagation	✗
R10	forward traceability	✗
R11	backward traceability	✗

Table 6. Ranking of application-centric architectures versus composition and traceability requirements

Official and high-end semi-official applications are few, well-known, typically listed in corporate and organization homepages and portals, and often part of the "training" a newcomer receives. They are shared by the entire organization, which is both their reason of being and the cause of their main weaknesses. Awareness of the existence of these applications and the data they contain is thus high. We can consider that official applications satisfy both requirements **R12** (sharing) and **R13** (awareness).

Shadow applications on the other hand are by definition circumscribed to the actor owning it. When the owner considers his shadow application of interest to a wider group and mature enough, he typically does not have a central location to advertise it, failing to satisfy requirement R12 (sharing). An actor with a new requirement which is already implemented in a shadow application elsewhere in the corporation has no way of finding it. Requirement R13 (awareness) is thus not met by shadow applications.

Neither official nor shadow applications provide a reliable way to determine relevance for a given user beyond word of mouth, failing to satisfy **R14** (relevance).

requirement		official applications	shadow applications	application-centric architecture
R12	sharing	☑	✗	✗
R13	awareness	☑	✗	✗
R14	relevance	✗	✗	✗

Table 7. Ranking of application-centric architectures versus collaboration requirements

Information system governance typically covers only the official applications, ignoring shadow applications. Considering the relative numbers of official and shadow applications, we can

consider that an application-centric information system is ungoverned and ungovernable, and that **R15** (governability) is not met.

requirement		official applications	shadow applications	application-centric architecture
R15	governability	☑	✕	✕

Table 8. Ranking of application-centric architectures versus governance requirements

4.2.3. Summary

Application-centric architectures are the most common information system scenario today, where official applications provide consistent and robust but rigid record-keeping, and shadow applications provide the flexibility actors need in their daily work, compensating for the lack of agility of official applications.

As a whole, we consider that application-centric architectures have sacrificed consistency and governability to provide the necessary business agility. The next section describes and evaluates the more recent service-oriented architecture paradigm.

4.3. Service-Oriented Architectures

The term Service-Oriented Architecture (SOA) can designate a great variety of solutions. The next section describes the assumptions we have made, most importantly the emerging view that *services* and *mashups* are complementary technologies [33, 35]. We then evaluate the SOA paradigm in general versus our proposed requirements for an agile information system.

4.3.1. Description

The Service-Oriented Architecture or SOA paradigm has been under discussion since the late 90s [65], but lacks a generally accepted definition [66]. A Service-Oriented Architecture views the corporate information system as a set of *services* which allow different applications to exchange and process data [67]. Desirable properties usually include loose coupling, dynamic binding, published interfaces and standardized technologies [33].

The basic principles of loose coupling and dynamic binding are illustrated by the familiar triangular diagram below, which shows how the service registry provides a level of indirection (i.e. decoupling) between the service consumer and provider [68].

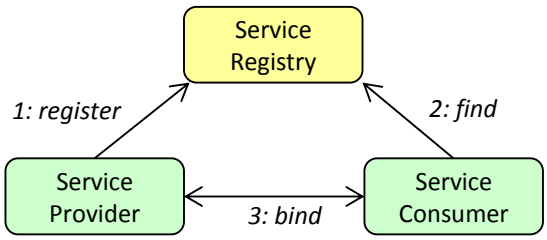


Figure 4-4. The SOA triangle, ensuring loose coupling between consumers and providers

Enterprise mashups put a visual face to the purely machine-to-machine services of SOA [69, 70, 71, 35] and can be considered an evolution of SOA [72]. Like services, mashups lack a commonly accepted definition[9]. They aim at allowing end users to integrate and combine services, data and other content [73] to bridge the business/IT gap [33]. The figure below presents the service-centric scenario which we will use in our evaluation.

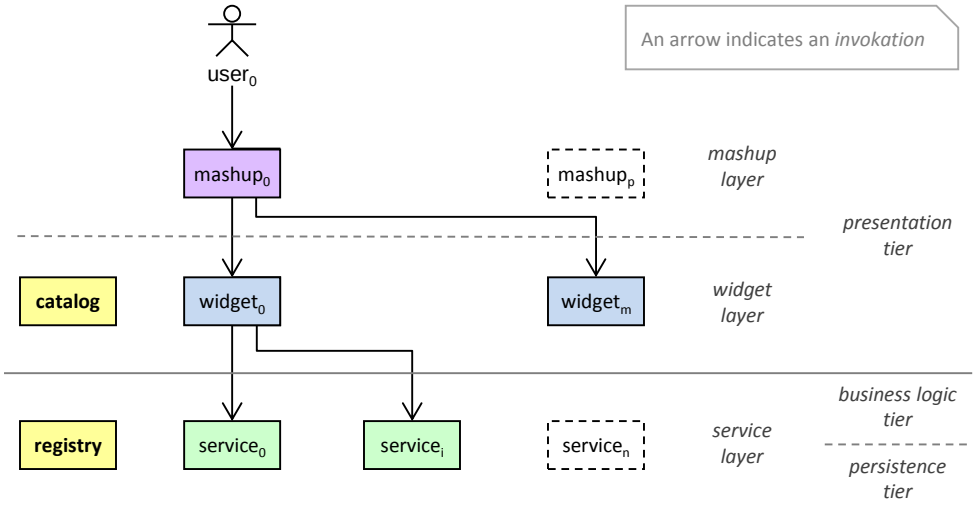


Figure 4-5. Service-centric architecture

At the bottom it shows a service layer, which exposes all available business data and functionality as standard and composable services. A registry allows both services to find each other and actors to find services they can use in *compositions* (not represented), which can be hard-coded, or use orchestration [74], data mashups [73] which can be considered "user-driven micro-orchestration" [75], or other mechanisms.

Widgets are small, configurable user interface snippets, which once bound to services provide a way for end users to interact with business data and functionality. They follow certain standards [76] allowing them to be manipulated in presentation-level mashup environments [33], where end-users can search the catalog for interesting widgets, and then select, compose and configure them to form an optimal user interface, tailored and configured for their specific needs.

We thus define a service-centric information system as a scenario where business requirements are implemented in services, which are complemented by widgets for user interaction.

definition **Service-centric Information System**
Information system where requirements are implemented as services with the associated widgets.

[9] In [73], 16 different definitions have been identified

Web services [77] are presently the most common solution for implementing an SOA [78], and we will thus assume that communication between the components of a service-centric information system is *synchronous*.

In the next section we evaluate this service-centric information system scenario versus our requirements for an agile information system.

4.3.2. Evaluation

When actor A faces a new business requirement, this can impact a service, a widget, or both. It is generally possible for actor A to implement his adaptation in a new service or widget, either by *substitution* or *encapsulation*.

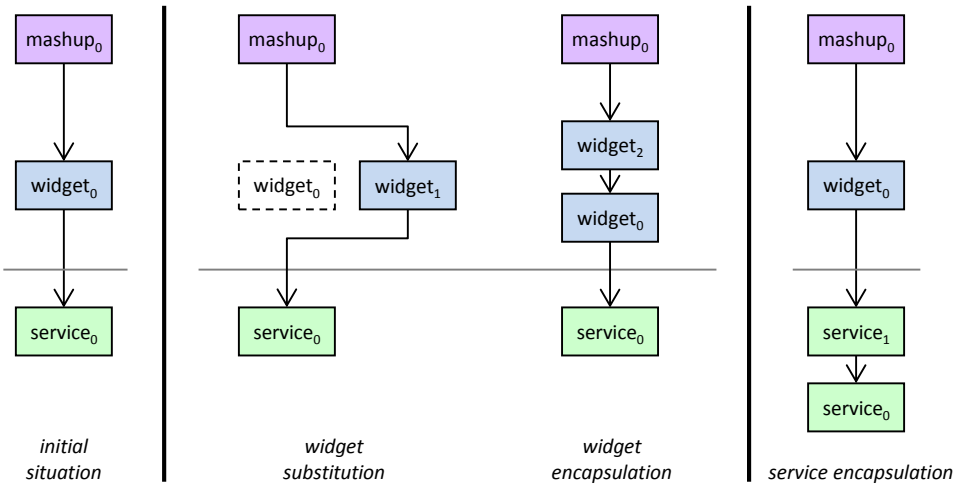


Figure 4-6. Widgets and service substitution

In an ideal substitution scenario, $widget_1$ reuses the code from $widget_0$ and implements only the difference. In the worst case, $widget_1$ is a complete re-implementation. In the encapsulation scenario, $widget_2$ *reuses* an instance of $widget_0$, pre-processing its input and post-processing its output, for example to add or remove a column in a table or to transform certain data or presentation elements.

If the underlying service $service_0$ is impacted, a new service $service_1$ must be developed. $Service_1$ can provide additional or substitute business logic, local storage of additional data elements, and invoke $service_0$ when required. This implies a mechanism for $widget_0$ to determine which service it should invoke depending on the actor, which can be dynamic (a service registry as illustrated in Figure 4-4) or static (deployment-time widget configuration).

Thanks to their relatively fine granularity, services and widgets can thus be adapted through encapsulation or substitution when necessary, which satisfies both requirements **R0** (influence) and **R1** (isolation). Since such adaptations can run on business-unit owned hardware, **R2** (hardware independence) is satisfied as well.

Business process modeling languages and techniques [79] allow in theory to involve domain experts in service composition and orchestration. Graphical data mashup environments like Yahoo/pipes [80] allow users to visually connect and configure services to adapt them to their needs. The development of composite services can thus be considered fairly accessible to end

users. However the development of "leaf" services still requires appropriate software engineering skills to deal with transactions, multi-threading, authentication and authorization, persistence, etc.

Widgets are user interface fragments, and like bigger user interfaces are often hardcoded. However, specific tools [80] provide a way for end users to graphically build widgets and bind these to the appropriate services.

While the development of very specific services and widgets requires software development expertise, the availability of graphical development environments partially satisfies requirement **R3** (ease-of-modification).

Actors can introduce specific services and widgets, but have no obligation to publish these in the respective registries and catalogs, thus hiding their adaptations from other actors. Such security through obscurity [81] can be sufficient in a corporate environment. For situations where it is not sufficient, the presence of authentication and authorization mechanisms in service environments allows actors to restrict access to their adaptations, satisfying requirement **R4** (confidentiality).

The table below summarizes the rankings of service-centric architectures versus application-level requirements R0-R4.

requirement		service	widget	service-centric architecture
R0	influence	☑	☑	☑
R1	isolation	☑	☑	☑
R2	hardware independence	☑	☑	☑
R3	ease-of-modification	✗	☑	☑
R4	confidentiality	☑	☑	☑

Table 9. Ranking of service-centric architectures versus application-level requirements

Services provide an ideal single point of access for reference data, as well as a mechanism to expose reusable business or technical functionality. In a service-centric architecture, it is natural to reuse existing services, removing the need for data replication which causes inconsistency in application-centric environments. Services are often used to update various underlying legacy applications in one call, hiding the replication. Services satisfy requirement **R5** (consistency).

The downside of this natural, synchronous way of reusing services is that they result in a tightly coupled network, with multiple runtime dependencies. Most services are thus not self-sufficient, and can be directly impacted by slow and unreliable dependencies. The underlying assumption in a service-centric scenario is that the other services are reachable and that the network is fast enough, which is a valid assumption in the restricted area of official and high-end semi-official services. When considering numerous services contributed by any actor anywhere in the corporation, the assumption breaks. While it is possible to design asynchronous service architectures, it is not the natural usage scenario and we thus consider

that service-centric architectures do not satisfy requirement **R6** (resilience) in an environment with many user-contributed services (R0-R3).

Mashups and widgets provide good support for unifying elements from various origins into profile-specific user interfaces. While the natural approach is to assemble widgets side-by-side, it is possible to interleave elements through the aforementioned graphical composition environments. A service-centric environment satisfies requirement **R7** (uniformity).

requirement		service	widget	service-centric architecture
R5	consistency	☑	-	☑
R6	resilience	✗	☑	✗
R7	uniformity	☑	☑	☑

Table 10. Ranking of service-centric architectures versus information system-level requirements

Services are well suited for composition. Service composition can be hard-coded or configurable, with high-level service composition languages like BPEL4WS and WSCI [82] or with graphical service composition interfaces.

Requirement **R8** (profile-driven composition) states that if several adaptations are relevant for a given actor A, these should automatically be combined for him. This is not met by the service layer, where service adaptation remains a manual operation. Likewise, if two separate widgets provide adaptations, there is no automatic way to apply both these adaptations for actor A. Only in the mashup layer can widgets be automatically inserted depending on the actors' profile. We can thus consider that requirement R8 (profile-driven composition) is not met, and that composition remains an essentially manual operation.

We can consider that a composed service encapsulates other services, and that it thus isolates its client from underlying changes. In the case of using service composition to merge adaptations, this fails to meet requirement **R9** (change propagation), which would dictate that changes in the underlying services get dynamically reflected in the top-level service.

Widgets are typically hard-coded. In the case of adaptation by substitution ($widget_1$), an evolution of the initial widget must be manually propagated to all adaptations. In the case of adaptation by encapsulation ($widget_2$), the adapted widget will dynamically reflect the change only if the evolution of the initial widget does not break the post-processing mechanism. We thus consider that requirement **R9** (change propagation) is not met.

In layered architectures, lower-level elements typically don't know their clients in higher layers. Services thus don't necessarily know by whom they are encapsulated or otherwise used. Likewise, widgets which have been adapted by substitution or encapsulation do not provide their owner with a reference to the various adaptations. Service-centric information systems thus do not meet requirement **R10** (forward traceability).

One of the main purposes of services, whether adaptations or not, is to hide whatever is underneath. This is in contradiction with requirement **R11** (backward traceability), which states that users are interested in the origin of the data they manipulate. Widgets may or may

not expose their data sources (i.e. associated services) as configuration parameters, possibly providing some insight. The origin of the widget itself is usually known through its URL, but adaptations break this. We can thus consider that no traceability mechanism is available, failing to satisfy requirement R11 (backward traceability).

requirement		service-centric architecture
R8	profile-driven composition	✗
R9	change propagation	✗
R10	forward traceability	✗
R11	backward traceability	✗

Table 11. Ranking of service-centric architectures versus collaboration and traceability requirements

In a service-centric architecture scenario, two components clearly meet requirement **R12** (sharing). The central service registry and widget catalog provide actors willing to make their adaptations available to a wider group with a central location where respectively services and widgets can be published, described with the appropriate metadata. Whether these referential systems are indeed used and searchable is questionable [68, 83] and depends on the relevance of the metadata, but the mechanism for satisfying requirement **R13** (awareness) is available. There are no indicators beyond owner-provided metadata to determine how relevant the various candidate elements are for an actor who is searching, failing to meet requirement **R14** (relevance).

requirement		service	widget	service-centric architecture
R12	sharing	☑	☑	☑
R13	awareness	☑	☑	☑
R14	relevance	✗	✗	✗

Table 12. Ranking of service-centric architectures versus collaboration requirements

Although a service-centric architecture can provide good visibility of available elements through the service registry and widget catalog, the lack of traceability mechanisms makes it difficult to get a clear overview of the dependencies between components and thus of the information system as a whole. We will thus consider **R15** (governability) not satisfied.

requirement		service	widget	service-centric architecture
R15	governability	✗	✗	✗

Table 13. Ranking of application-centric architectures versus governance requirements

4.3.3. Summary

The enormous amount of research on service-oriented architectures precludes a complete state-of-the-art, and we have chosen to consider the emerging combination of services and enterprise mashups as representative of the potential of SOA.

The fine-grained nature of services and widgets in theory allows actors to substitute or encapsulate unsatisfactory elements, introducing a level of agility while preserving the overall consistency. Chains of synchronous elements however introduce a resilience concern, composition remains manual and traceability is not provided. And while (in theory) a service registry and widget catalog provide a centralized sharing mechanism, they provide no way to deal with potentially overwhelming numbers of actor-contributed elements.

Overall, the fine-grained and distributed nature of services and widgets seem to provide a better foundation for an agile information system than the application-centric scenario. But, in spite of some end-user graphical tools for end-users, they target software professionals, and provide no improvement in traceability and governability.

In addition to the two extreme architecture paradigms presented earlier, the next sections present various research domains which relate to business agility, either by aiming to cope with requirements complexity, to build agile systems, or to deal with the fragmentation of information systems.

4.4. Coping with Requirements Complexity

In a previous chapter we have identified organizational complexity and the associated requirements paradox as an obstacle for information system agility. This has been widely studied, and this section briefly presents a selection of research topics coping with multiple viewpoints during the various phases of the software lifecycle.

4.4.1. Requirements Engineering

Two decades of studies on *viewpoints* [84, 85] have focused on capturing and representing divergent concerns and reconciling these at the specification and design level. Conflicting requirements can thus be expressed during inception, but these must be solved before implementation. Viewpoints are thus helpful in both application-centric and service-centric scenarios. They cannot be accurately evaluated versus our requirements, but will be compared to our proposal in section 8.5.

With respect to managing conflicting requirements [86], research in requirements engineering is considered *"not really successful"* [87]. Recent work proposes to apply social mechanisms [46] to reduce both the requirements and knowledge/influence paradoxes by allowing greater numbers of actors to vote and comment on requirements. Our proposal will build upon a similar principle.

4.4.2. Model-Driven Engineering

Model-Driven Engineering (MDE) [88, 89] elevates the level of abstraction at which software is developed, turning models into central and productive artifacts. By automating the production

of business applications, MDE can significantly increase agility by reducing the time and effort required to implement new requirements.

The *Software Language Engineering* [16] and *Domain-Specific Languages* [9] 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, which also leave room for imperative aspects.

Model-driven engineering thus provides foundation concepts for a significant reduction of the knowledge/influence paradox, which our proposal builds upon.

4.4.3. End-User Software Development

The *End-User Software Development* (EUSD) or *End-User Programming* (EUP) community denounces the fact that the role of humans who will use the system is marginalized to that of *"a source in requirements elicitation or worse, a "user", instead of being considered a free and intelligent agent of change"* [90, 15]. EUSD advocates the empowerment of end-users to implement their own specific requirements, as a way to bridge the business-IT gap [56, 91].

In the context of business applications, spreadsheets have been intensively studied [21, 26], as well as enterprise mashups [33, 73, 35, 92] and more recently collaborative and social aspects in enterprise settings [93].

In enterprise information systems, we subscribe to the view that end-users are the ultimate domain expert, and our proposal will apply EUSD principles to information system evolution.

4.4.4. Software Composition

At the implementation or programming level, *Subject-Oriented Programming* (SOP) [94] questions the possibility of a global concept of class, and advocates the decomposition of software into multiple *design subjects*. The main objectives are to avoid tangling different and potentially conflicting requirements, reducing the monolithic nature of a design, and allowing for concurrent development [34].

This initiative appears to have merged with the aspect-oriented programming (AOP) [95] domain which has brought unquestionable benefits in the simplification of enterprise component development [96] but appears to have lost its promising focus on subjectivity along the road.

4.5. Achieving Business Agility

4.5.1. Shadow Applications

Shadow applications are a widely known but widely accepted problem. They are frequently mentioned when discussing information system agility [60] and studying dissatisfaction with business applications [29], but not necessarily considered as a problem which must or can be addressed [39].

4.5.2. Agile Methodologies

The 1994 "Chaos Report" [97] reports the shocking rate of only 16% of successful IT development projects[10], attributed to heavyweight plans, specifications and other documentation imposed by maturity models and process compliance [20] considered incompatible with the accelerating pace of change. In response, various alternative software development methodologies like eXtreme Programming (XP) [98] and SCRUM [99] have emerged, which after agreeing on a set of common principles [100] are now commonly referred to as *agile* methodologies.

Through their emphasis on short iterations, working software, test automation and end-user involvement [51, 100], agile methodologies have been embracing continuous change as the norm for software products and aim at participating in business agility.

However, agile methodologies do not provide the expected benefits for larger projects [20], and empirical studies show that the theoretically sound principle of prioritization driven by business value leaves (up to 90% of) features unimplemented [101], with no other choice for actors who really need these features than to implement them themselves in shadow applications. Besides, by facilitating success of smaller projects we think agile methodologies actually aggravate shadow application proliferation.

4.5.3. Software Tailoring

The *tailoring* of enterprise systems, from simple configuration to the modification of commercial code, is a topic of sufficient complexity for [26] to propose a typology of its types. The tailoring community advocates that software in general needs to be as flexible as possible, and in the case of information systems that the ability to adapt to a continuously changing environment is undermined by the "fallacy of 'correct' information systems requirement specification" and should be approached as a key factor of success [102, 103]. To achieve this, applications should try to be *ateleological* [104], i.e. independent of a specific end. Like the end-user software development community, they thus imply the transfer of responsibility from application developers and designers to application users.

4.5.4. Cloud Computing

The evolution of cloud computing from infrastructure-as-a-service (IaaS) to software-as-a-service (Saas) [105] has recently yielded research in *multi-tenancy* [106], a way to configure the same software installation for different corporations. The goal is to support different corporations and thus different requirements on the same installed business application, which necessitates an implementation supporting isolation (R1) and confidentiality (R4). Beyond this commonality, we think supporting different corporations is a different problem, and cannot position multi-tenancy versus the rest of our requirements.

[10] The "Chaos Report" and subsequent Standish Group reports, although questionable in their definition of failure and heavily criticized [171], remain a cornerstone of agile culture.

4.6. Coping with Information System Fragmentation

In previous sections, we have presented the fragmentation of corporate information systems over a huge number of applications, most of them shadow applications. We can distinguish between the following kinds of fragmentation.

- *Historical* fragmentation is a consequence of mergers and acquisitions. Considering the cost and risk of disruptions involved in aligning working organizations (the acquiring and the acquired corporation) with foreign processes and applications, it is common for the respective information systems to live side-by-side for an extended period of time.
- *Accidental* fragmentation occurs when an actor has a requirement, is not aware of the application meeting it, and thus decides to introduce a new application.
- *Intentional* fragmentation occurs when an actor has a requirement which is met by no existing application, and he decides to introduce a new application.

Various research fields and technologies aim at helping corporations cope with all the above kinds of fragmentation, after the fact. We can distinguish at least the following domains.

4.6.1. Enterprise Application Integration

Enterprise Application Integration (EAI) aims at connecting applications together. [107] proposes 4 levels at which integration can be applied: the data level (persistence tier), the application interface level and method level (business logic tier), and the user interface level (presentation tier). EAI products typically provide a message-oriented-middleware (MOM) core, with pre-developed configurable adaptors for most popular enterprise packages and standard technologies (SQL, SOAP...). *Enterprise Service Buses* (ESBs) are an evolution of EAI [108].

The difficult part of integration is not technical, i.e. reliably moving data from one environment to another, but dealing with the *semantic heterogeneity*, which implies a translation of the data from the source system into the proper equivalents in the destination systems [109]. This can involve complex transformations and mapping mechanisms, not always fully automated. *Enterprise Ontologies* can provide a neutral domain-specific model to act as a pivot in these transformations [110].

It is debatable whether these technologies are beneficial to overall business agility. On one hand, they help in connecting applications and services, minimizing or even hiding the inconveniences of fragmentation. On the other hand, they may encourage coupling of applications, fairly loose from a technical standpoint but no so much at the semantic level. They also represent yet another piece of the puzzle which yield discussions, risk and update costs when envisioning a change.

4.6.2. Business Intelligence

The field of *Business Intelligence* (BI), an evolution of mere reporting, aims at building *data warehouses* which incorporate all available enterprise data in one database, usually via Extract-Transform-Load (ETL) technologies [111]. Closely related, the goals of *enterprise information integration* (EII) and *federation* approaches are to provide a uniform read-only

access to multiple data sources without having to first load them into a data warehouse [112], by building a global schema which can be queried by users or applications.

4.7. Conclusion

In this chapter we have described two information system architectures, representing the extremes of a continuum of possible scenarios. We have evaluated them with respect to our research hypotheses R0-R14. The table below summarizes the marks of both scenarios.

Category	Requirements	Application-centric architecture	Service-centric architecture
Application	**R0**: Influence	☑	☑
	R1: Isolation	☑	☑
	R2: Hardware Independence	☑	☑
	R3: Ease of Modification	☑	☑
	R4: Confidentiality	☑	☑
Information System	**R5**: Consistency	✗	☑
	R6: Resilience	☑	✗
	R7: Uniformity	☑	☑
Composition	**R8**: Profile-driven Composition	✗	✗
	R9: Change Propagation	✗	✗
Traceability	**R10**: Forward Traceability	✗	✗
	R11: Backward Traceability	✗	✗
Collaboration	**R12**: Sharing	✗	☑
	R13: Awareness	✗	☑
	R14: Relevance	✗	✗
Governance	**R15**: Governability	✗	✗

Table 14. Summary ranking of application- and service-centric architectures

We have presented various research domains related to business agility, which we will build upon in our proposal for an alternative enterprise architecture presented in the next chapter.

5. Social Information Systems

The previous chapters have illustrated that present information system paradigms do not provide an adequate level of business agility. Our objective is to define an enterprise architecture principle meeting all our requirements for an agile information.

This chapter describes the principles guiding our proposal. We first propose an **alternative decomposition** of business applications, splitting application elements into smaller fragments and then re-composing truly profile-specific applications. We describe how the **distributed ownership** of these fragments can provide a high level of business agility, and how social technologies can be applied to share them across the corporation and achieve good levels of awareness and **governability**.

These principles will be refined in chapter 6, which describes a possible architecture, and refined further in chapter 7 which presents our prototype implementation.

5.1. An Alternative Decomposition

Present architecture paradigms assume a single, consistent, *objective* view of the way a corporation works, and thus of its information system.

definition **Objective**
adjective: of, relating to, or being an object, phenomenon, or condition in the realm of sensible experience *independent of individual thought and perceptible by all observers* [113].

We consider shadow applications as evidence that such an objective view is impossible, and think that each actor needs his own *subjective* view of the corporation and its information system.

definition **Subjective**
adjective: characteristic of or belonging to reality *as perceived rather than as independent of mind* [113].

In theory, subjectivity could be achieved by a central set of elements with the appropriate *filtering* mechanisms. Indeed, such role-driven filtering mechanisms are present in a majority

of business applications. They assume the existence of a super-element, objective, complete and consistent, from which pieces get *subtracted* according to the actors' profile as illustrated by the figure below.

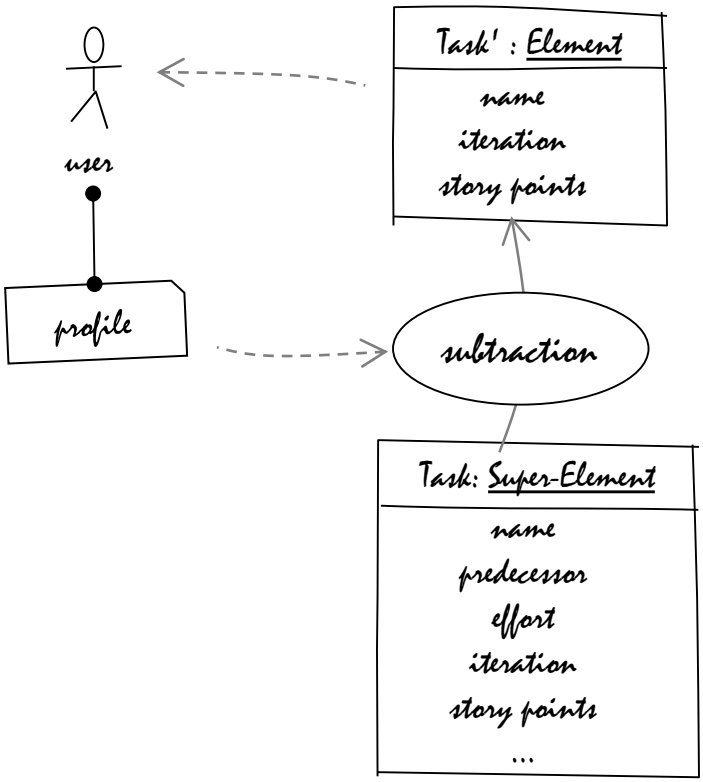


Figure 5-1. Subjectivity through subtraction

We think this assumption is the key problem with present business applications. The timely design of super-elements is precisely the impossible feat we have described in section 2. The above example deliberately includes mutually exclusive attributes (*predecessor* and *effort*, used for traditional project management [114], and *iteration* and *story-points* used in agile methodologies [100]) to illustrate the tension[11] between objectivity and consistency.

In this section we propose an alternative view on business computing, decomposing application elements into *fragments*, applications into *perspectives* and then using the users' profile to automatically recompose a *profile-specific* application.

5.1.1. Element De-Composition: Fragments

Instead of building an objective super-element (i.e. the union of all subjective elements), we envision building a set of sub-elements which we will call *fragments*. Fragments will be described in depth in the next sections, but at this stage it is sufficient to state that fragments are the constituents of application elements.

[11] It could appear natural to use inheritance to represent mutually exclusive attributes, but the number of profiles, i.e. combinations of groups (see Figure 2-3), precludes this.

definition **Fragment**
Part of an application element, either standalone or
an adaptation which can be applied to another Fragment.

Instead of a subtraction mechanism *removing* pieces, we propose a composition mechanism *adding* fragments, as illustrated by the figure below.

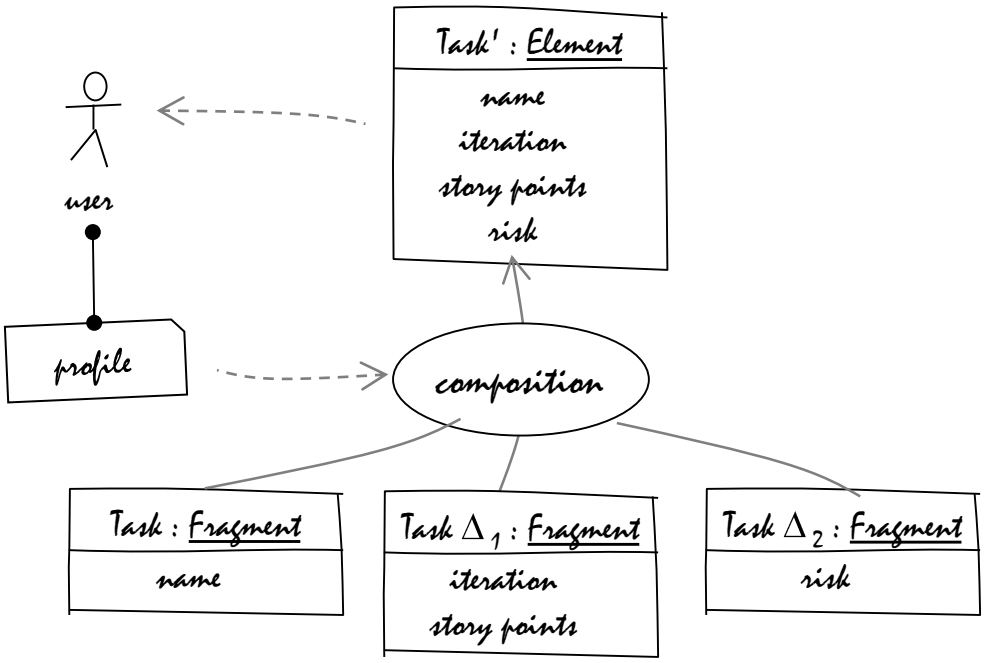


Figure 5-2. Subjectivity through composition

Instead of designing a super-element, i.e. the frozen union of distorted and incomplete requirements, a composition scenario implies designing fragments, i.e. the various *intersections* of requirements from all actors, with a (probably small) common root fragment which all actors agree upon.

We think a composition approach presents a number of important benefits over subtraction.

- As illustrated in Figure 2-17, reaching an agreement on a super-element in big corporations involves a number of hurdles. Smaller fragments are significantly easier to agree upon, especially since fewer actors are involved.
- Due to the high number of possible profiles in big corporations, we cannot expect super-elements to contain all possible useful attributes. In particular, *individual* attributes like the “risk” attribute above are not realistic in a subtraction scenario. In a composition scenario, it becomes possible to compose *profile-specific* applications, where even an individual requirement represents just one more fragment to add to the final element.
- Finally, a central super-element implies a central owner and thus a potential bottleneck. In contrast, multiple fragments enable to distribute the ownership to the most knowledgeable actors, which we will discuss in section 5.2.

Considering the requirements paradox presented in section 2.5.1, the bigger the corporation the smaller the first level of intersections are likely to be, to the point of being of no practical use to anybody. Their purpose is to act as a *scaffold* which other actors can *add* their elements to, at business unit level, group level, or even individual level.

At subsequent levels of the organization, this process can be repeated recursively, defining successive layers of fragments representing *boundary objects* [115]. Boundary objects allow actors with different interests to collaborate around entities despite the fact that they attach different semantics to it. They provide a form of local agreement [116] enabling collaboration.

Having decomposed super-elements into fragments, the next section presents how to decompose applications into more subjective constructs.

5.1.2. Application De-Composition: Perspectives

Present applications attempt to cover a more or less broad business domain, again aiming at objectivity. Instead of grouping fragments per domain, we propose to group them per subjective viewpoint of a given actor on the information system. We call such viewpoints *perspectives*.

definition **Perspective**
Subjective viewpoint of a given actor on the information system.

A perspective hosts all relevant fragments for a given actor. Reverting back to our running example, the figure below illustrates how the “Luxury” perspective defines the root fragment for business concept “Request”, and the “quality group” perspective defines an extension. Both perspectives host the *model* (concepts, attributes, relationships, rules) and the associated *instances* (actual business entities).

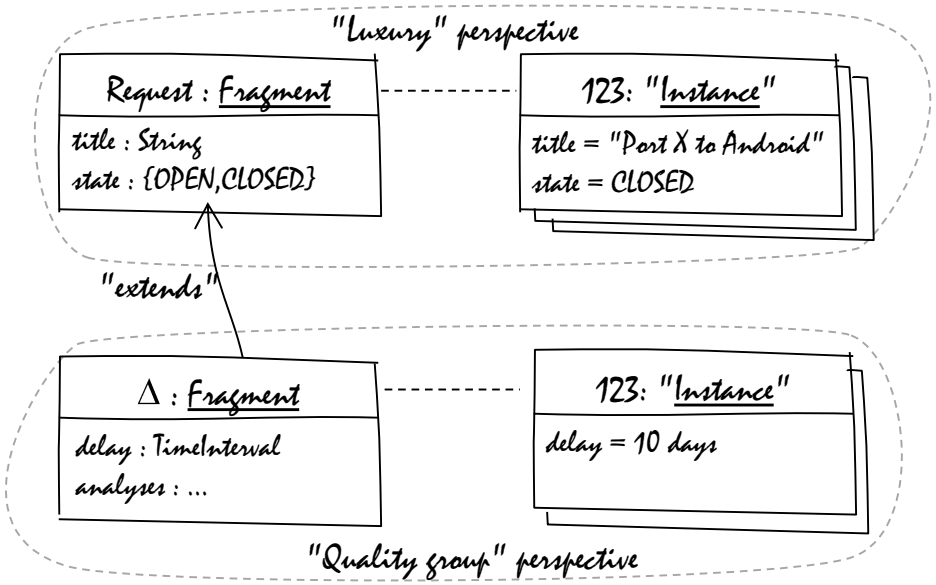


Figure 5-3. Perspectives hosting fragments and instances

As a first description of perspectives, we can say they must provide the following primitive capabilities.

- define a new fragment
- extend an existing fragment

Furthermore, in order to represent the viewpoint of a given actor on the information system, perspectives must also indicate which fragments from other perspectives are relevant. We call this primitive operation *import*.

- import a fragment from another perspective

If we make the assumption that all actors define one and only one perspective, the example individual “Maria” in Figure 2-3 inherits the following set of perspectives from the various groups she is a member of.

```
perspectives(Maria) = { organization/manufacturing,
                        region/Europe,
                        region/Europe/France,
                        region/Europe/France/Grenoble,
                        role/quality,
                        project/Beta,
                        community/agile,
                        users/~maria }
```

Figure 5-4. Perspectives of individual Maria

An actors’ full set of fragments thus contains the fragments defined in his own perspective ("users/~maria" in the example), but also those defined in perspectives belonging to groups he is a member of. We will call the latter *inherited* fragments.

It is thus possible that a given actor inherits fragments which he is not interested in. In this case, he should be able to indicate this lack of relevance in his own perspective, which dictates our last primitive capability.

- unimport an inherited fragment

The figure below shows the meta-model representing the concepts presented so far, and introduces the terms *directory*, the referential server for actors, and *repositories*, servers hosting perspectives and their fragments.

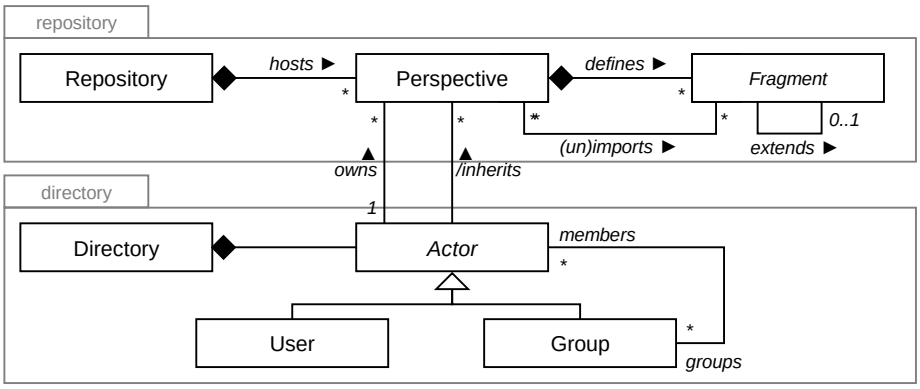


Figure 5-5. High-level meta-model of Perspectives and Fragments

Perspectives are the fundamental concept underlying our proposal. They intend to unify the notions of a COTS application, its configuration and customization, personal preferences, the surrounding shadow applications and to some extent even present integration mechanisms.

Fragments represent different, finer and more connected information system grains than applications. As such, they can be composed to form profile-specific applications, as described in the next section.

5.1.3. Profile-Driven Re-Composition

In previous sections we have shown the high number of possible profiles in big corporations, and how in present information systems this translates into a high number of shadow applications.

A major benefit of decomposing information systems into fragments and perspectives is that it enables the construction of *profile-specific* applications. Given the current users' profile (i.e. set of groups), we can traverse the graph of groups to find all relevant perspectives, and thus all relevant fragments. This set of relevant fragments can be composed to form an application tailored for the current user, containing everything relevant for his specific profile and nothing superfluous. Considering our goal of maximum agility and in order to avoid premature optimization [117], we envision this composition happening at run-time.

The diagram below shows the previous meta-model augmented with the runtime concepts of profile-specific application and element.

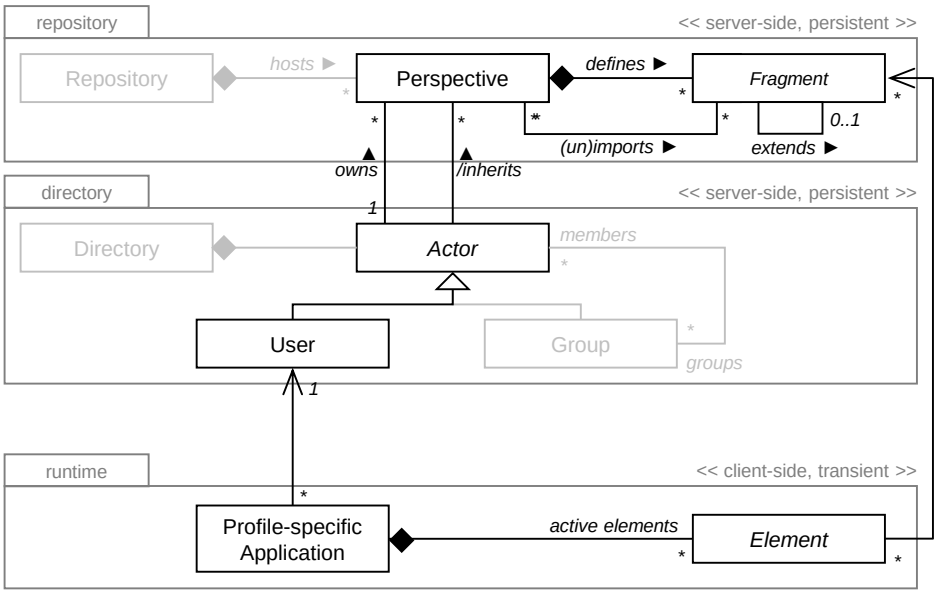


Figure 5-6. Runtime meta-model of profile-specific applications

The composition logic allowing to weave fragments together to form profile-specific elements in a resilient manner is not trivial in a distributed environment and will be described in section 6.2.1. Once these elements are available it is possible to build a profile-specific user interface with forms containing only the concepts and attributes relevant for the current user, as illustrated below and as described in further detail in section 6.2.2.

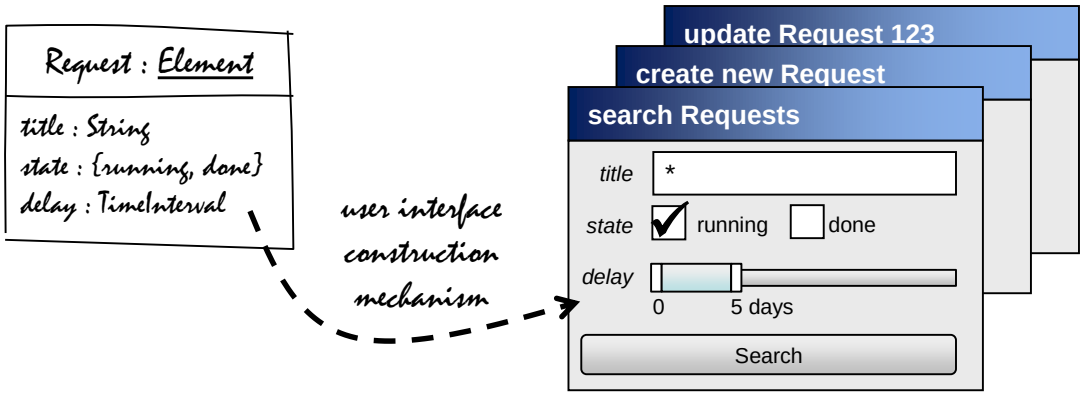


Figure 5-7. Example element and associated presentation layer

5.1.4. Summary

Our definition of the term application encompasses 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 root fragments to tiny individual perspectives with just a few adaptations replacing simple spreadsheets. Some perspectives may only factorize the optimal list of import and unimport declarations for a given actor.

Perspectives and fragments in essence provide a mechanism to partition the information system along multiple dimensions, acknowledging and anticipating the fact that actors have different requirements and need to be isolated from each other to be agile. Perspectives thus allow to defuse the requirements paradox, and to compose profile-specific applications, tailored to include all relevant fragments for the current user.

In the next section we describe how perspectives provide a way to solve the knowledge/influence paradox as well.

5.2. An Alternative Distribution of Responsibilities

We have shown through shadow applications that the ownership of information system elements is not centralized today. Likewise, the ownership of the aforementioned perspectives and fragments should be distributed to the right actors, as advocated by aforementioned research on end-user software development and information system tailoring.

Ideally, perspectives should be owned by the "most knowledgeable" actor, “closest” to the business concern at hand. These criteria are subjective, and in most organizations controversial or even provocative. However, through the isolation mechanism which perspectives provide, in case of disagreement multiple actors can be considered most knowledgeable in their particular domain. We do make the assumption that actors behave responsibly, adding only fragments which they really need and which don't already exist as per their knowledge.

Such user-contributed fragments can be *shared* with other actors. This effectively allows all actors to contribute to the information system, which raises the question of how to cope with these potentially great numbers of fragments emerging from the bottom up. Fortunately, this question has been answered in a more complex environment with less disciplined contributors which is the consumer space.

In the last decade, the consumer web has moved away from centrally controlled content to user-contributed content [118]. Whether the user community contributes pictures on flickr[12], videos on youtube[13] or dailymotion[14], or other media, the problem of coping with a huge and ever-increasing mass of elements has been addressed by leveraging the collective energy of the systems’ users [119] through various *social* mechanisms.

In the following sections we describe our vision transposing the consumer-space user-generated content trend and the associated self-organization mechanisms to perspectives and fragments in a business setting.

5.2.1. User-Contributed Fragments

Organizational hierarchy is a natural and convenient way of propagating fragments from the top to the bottom of a corporation (see Figure 2-3). However, due to the knowledge/influence paradox in particular, many important fragments can only be created at much lower levels.

The fact that a given actor defines a fragment does not necessarily mean that it is relevant only for him. Indeed, few problems are completely specific to one actor and thus most solutions, even (or especially) when initiated “in the field”, are of potential interest to other actors of the corporation, close by or far away. It is thus possible for any actor to define a fragment of

[12] www.flickr.com
[13] www.youtube.com
[14] www.dailymotion.com

potential interest to a wider community. This may be part of the mission of the actor, for example a software quality group providing fragments related to software root cause analyses. It can also be accidental, in the case of an individual who solves a problem for himself but which he afterwards realizes may occur in other places in the corporation.

In order to encourage re-use of user-contributed fragments, a complementary cross-organization *fragment sharing* mechanism is required. Fragment owners must be able to indicate whom they want to share their fragment with. This could be the colleague in the next cubicle, his project team, a given community or even the entire corporation.

In a similar fashion to service registries and widget catalogs, these indications must be published in a central location. We call this operation *export*, the symmetrical operation from the import operation presented previously. We thus need one final primitive operation for perspectives.

- export a fragment to other perspectives

Import and export operations can be specialized according to the nature of the fragment, as we will describe in section 6.3.

The implication of this sharing mechanism is that everybody becomes a potential provider of information system fragments, sharing with just one colleague or with the entire corporation. This effectively shifts a number of formerly central responsibilities (most importantly developer, domain expert and administrator) *from* the corporate IT authorities *to* all other actors of the corporation (formerly "end-users").

Spreadsheets have been successful precisely because they have shifted power from the programmers to the end users [40]. Usability is a key success factor in such shifts, and we think that the relative simplicity and stability of the core concepts of data-centric business applications (see section 2) allows to envision a similar level of intuitiveness. While the dream of business software without programmers is at least as old as the COBOL language, even in recent research [92] this is still considered a radical paradigm shift.

There is no guarantee that people would exercise this power in the business application realm, but the success of spreadsheets, the number of shadow applications, and prior field studies are encouraging. In a recent survey involving 73 users of enterprise applications [16], 87% of participants have rated the potential benefit to ease or speed up their work to be at least 2 on a scale ranging from 0 (no benefit) to 4 (high benefit). 80% of participants would accept learning efforts of several hours up to several days to be able to create applications themselves. In another study involving 15 industrial participants, *"all users liked to develop their own software applications that suit their needs and interests"* [120].

Our assumption is that removing the bottlenecks and deadlocks caused by central super-elements could help liberate the *collective intelligence* [121] of the corporation. A dynamic corporation with skilled and motivated actors can thus expect to see thousands of shared fragments provided by its business units and employee base, which dictates the need for mechanisms to cope with these volumes presented in the next sections.

5.2.2. Classifying User-Contributed Fragments

Most social resource sharing systems use a kind of lightweight knowledge representation, called *folksonomy* [122]. Folksonomies rely on emergent semantics [123], which result from the converging uses of the same vocabulary by a large number of non-expert users. The success of these systems shows that they are able to overcome the knowledge acquisition bottleneck [122], i.e. the prohibitive time required to classify user-contributed items. Besides the items' name and description, folksonomy-based systems typically allow both contributors and end-users to *tag* items, i.e. attaching free-text keywords to items, often with sophisticated suggestion mechanisms [124].

In an enterprise setting, we envision folksonomies be applied to organize fragments. As an example, if a Luxury extension called "delay analysis" defined in a "planning" perspective is annotated with a simple tag "RCA[15]", this provides important semantic information for people in the quality community.

Folksonomies can be leveraged by various search mechanisms: text-based search similar to web search engines, tag clouds, and more traditional browsing of a hierarchical classification [122].

5.2.3. Estimating Fragment Relevance

One could consider that a big number of fragments is not a problem per se, but only becomes a problem when an actual actor needs to find something, hence requirement R14 (relevance). Another consumer-space social technology which should prove of great help is the *recommender system* paradigm.

Recommender systems have revolutionized the marketing and delivery of a variety of complex product offerings by providing personalized recommendations [125]. They build upon extensive research in cognitive science, approximation theory, information retrieval, forecasting theories, management science and also to the consumer choice modeling in marketing [126]. The recommendation problem can be summarized as the problem of estimating *ratings* for "new" items, i.e. which have not yet been seen by a user. *Collaborative* recommender systems [127] try to predict the rating of items for a particular user based on the items previously rated by other users, weighted by the similarity of the user profiles.

In the consumer space, profile matching is based mainly on data provided by the user (age, gender, education...), his activity (for example items bought) and his ratings [126]. In a perspective-centric information system, many data items are available to help determining profile similarity.

- *Group memberships* provide profile data along various dimensions (organization, role, projects, region...). Interestingly, in a corporate setting this data is usually available the first day, thus solving the "new user" or "cold start" problem

[15] Root-Cause Analysis

plaguing many commercial recommender systems who cannot determine the profile of newcomers [125, 128]

- Users with similar group profiles automatically inherit the same set of fragments. The set of additional fragments they choose to *import* is very similar to the list of items purchased in a commercial setting, i.e. a fairly accurate reflection of the actors' interests
- Another strong profile indicator is the set of fragments they choose to *unimport*, which can be interpreted as a rating of "not useful"
- Aforementioned *tags*, if similar, can indicate that different actors have the same viewpoint
- *Explicit rating* of fragments is possible as well, if it represents a low-effort operation for the end-user[16]. While probably odd to the present working population, rating the fragments in their workspace may appear entirely natural to the upcoming generation of knowledge workers who have grown up with such mechanisms
- Beyond these persistent aspects, more dynamic aspects can be collected during *system usage* [129] as an additional similarity indicator. Two users spending most of their time on the same class of objects or even actual objects, or navigating with the same pattern in the application, present a potential similarity

Other social mechanisms can be envisioned, like explicitly indicating *trust* in another actor, not unlike the LinkedIn[17] "connection" or Facebook[18] "friend" relationships, which could provide a weight to ratings where profile similarity is not sufficient. An *explicit recommendation* mechanism is possible as well, a generalization of the common phenomenon of people forwarding each other interesting things through informal communication channels like email.

5.2.4. Managing Fragment Awareness

Awareness of available fragments is closely related to their classification and the ability to compute correct relevance ratings.

The cross-cutting nature of perspectives is an ideal channel for raising awareness. As an example, the "delay analysis" extension can originate in a "planning" perspective, but can be exported to a "quality" perspective, making it visible to a community likely to be interested in it. Additionally, the social mechanisms described in section 5.2.2 allow to further organize fragments while their users recognize and classify them, continuously improving the visibility of fragments to interested users.

While searching for fragments, the ranking provided by recommender technology described in section 5.2.3 allows to present the most relevant results first, which is critical for awareness. More proactively, we can imagine (configurable) relevance thresholds above which users get notified of the existence of elements without searching for them.

---

[16] We refer to the "like", "+1" and other "thumbs up/down" buttons from social web sites
[17] www.linkedin.com
[18] www.facebook.com

5.2.5. Towards Social Information Systems

In the consumer space, the aforementioned social mechanisms have proven effective in organizing huge quantities of consumer-contributed data [119]. We think that a corporate environment, where all users are authenticated professionals, is an even more beneficial setting than the consumer space for social technologies to apply.

In an ideal situation, we envision *social information systems* where fragments are contributed from the bottom up, shared with other actors, adopted by some and improved through social feedback mechanisms, rated, recommended and if widely adopted eventually gradually "promoted" to more central perspectives. This could result in the *democratic* or *meritocratic* evolution of a corporations' application landscape, where the actual users decide which elements are the most useful for their daily jobs.

The aforementioned social technologies are not without issues. Especially regarding recommender systems, it is yet unclear whether these could create increasingly isolated sub-communities by making it easier for like-minded people to find each other [130] or on the contrary foster the (equally bad) opposite outcome, i.e. excessive homogenization. Regarding present recommender systems, some believe they help consumers discover new items and thus increase diversity, while others believe they only reinforce the domination of already popular items [131]. More research is needed to determine how such systems can be tuned to achieve the positive outcome we have described [125].

5.2.6. Summary

We have described our vision of a social information system, where all actors contribute[19] and share business application fragments, applying the vast array of social technologies from the consumer space to organize great numbers of user-contributed fragments at low cost, to determine their relevance for a given actor, and to manage awareness among concerned actors. We think this enables the collaborative design and social evolution of the corporate application landscape.

Allowing all actors to contribute to the information system represents a significant shift of responsibilities, from the corporate IT authorities to all other actors. Instead of providing complete solutions to all of the corporations' problems, a corporate IT department would merely provide an infrastructure, pre-populated with the right scaffolding fragments and then allow all actors to adapt it, which raises the question of *governance* of social information systems which we discuss in the next section.

5.3. Social Information System Governance

Due to the high percentage of shadow applications, only a small fraction of present information systems can be considered to really be governed. Social information systems leave fragments under the control of the community but host them in a unified infrastructure; the big difference with respect to governance is that this makes the situation observable, measurable, and thus manageable.

[19] This could be considered a form of internal *crowdsourcing* [143].

In this section we discuss three aspects of social information system governance: monitoring, community management and inconsistency management.

5.3.1. Monitoring

Monitoring essentially means collecting indicators to observe the evolution of a social information system and estimate its soundness. Once collected, indicators allow to compute trends, to express and periodically check consistency rules, and to build high-level dashboards, i.e. predefined ways of combining and presenting indicators.

Indicators can be collected by scanning the various repositories. A first type of indicators can be simple counters, like the number of defined root fragments, defined extension fragments, inherited fragments, imported fragments, unimported fragments, or export declarations. These can be computed per perspective, per actor, per server or for other dimensions. A possible use of such counters is to classify perspectives or actors. For example, the assertion below indicates a purely "provider" perspective.

```
isProvider(x):
  count(x.importedFragments)=0 AND
  count(x.definedRootFragments)=count(x.exportedRootFragments)
```

Figure 5-8. Example assertion for classifying Perspectives

Keeping the history of these counters allows observing the overall growth, computing averages and standard deviations, and possibly setting thresholds to warn about alarming trends applying statistical process control (SPC) [132] techniques.

It is possible to collect similar counters at fragment level. We can envision counting the number of adaptations, both in terms of fragments and attributes, measuring the "width" and "depth" of the set of its adaptations. Counting the number of times a fragment is imported, inherited and unimported, both in terms of perspectives and number of users, provides some insight in its impact.

Beyond the simple threshold and trend analysis mentioned above, rich data mining and pattern matching algorithms can be applied for a deep insight into the evolution of the information system. Indeed, the ability to compute these indicators and the visibility this provides are major expected benefits of our proposal.

5.3.2. Community Management

The social web is driven by *web communities*, which can be defined as *"a group of people who share a common purpose and interact with each other through a community platform"* [133]. Maintaining a high quality of user-generated content requires a healthy and thriving community (and vice versa), which has yielded recent research in community platform governance [134].

We think that employees contributing application fragments meet the definition of a web community, and that social information system governance will thus benefit from the advances in community platform governance research.

Beyond analyzing the contributions, the health of the community itself can be measured. Health of online communities is a fairly new and complex concept which is codependent on the emergence and evolution of user *behavior* [135]. A great variety of roles can be inferred by observing user behavioral patterns, like “popular initiator”, “elitist”, “over-rider” and “grunt”. Community health can then be estimated by observing the evolution of the balance between these various behaviors, enabling dashboards for a new enterprise role, the *community manager* (or community owner) [136].

Social information systems could thus provide a new angle on governance through community management, which should prove complimentary to the more traditional monitoring described in the previous section.

5.3.3. Inconsistency Management

Even though today's information systems are far from consistent, a spontaneous concern when envisioning an environment encouraging more individuals to contribute fragments is that it could *aggravate* inconsistency.

definition **Inconsistency**
Any situation in which two descriptions do not obey
some relationship that is prescribed to hold between them [137].

Inconsistency carries a stigma, implying poor quality work. However, in many cases it is desirable to tolerate or even encourage temporary inconsistency to facilitate distributed collaborative working [138], to ensure all stakeholder views are taken into account [139], to experiment with alternative solutions and to maximize reactivity. Inconsistency can be a driver of software evolution [140] as long as the associated risks are measured and periodically re-assessed [141].

Whether worse than the present situation or not, a social information system will certainly yield inconsistency and needs governance mechanisms to stay within the aforementioned range of beneficial inconsistency and avoid chaos. In this section, we present various mechanisms aiming at preventing, detecting, assessing and handling inconsistent fragments.

Preventing Inconsistencies "A Priori"

Although a social information system allows for a fully emergent model, large corporations are unlikely to start from scratch with an empty set of perspectives. In a real-life setting, the model would be bootstrapped with a number of scaffolding fragments like third-party applications, possibly ontologies (potentially also acquired from a third party) and certainly other core fragments decided by corporate authorities. Scaffolding is a form of implicit coordination through structure, which improves the quality of the result in the presence of many contributors [142]. By providing a common starting place for actors to introduce new fragments, scaffolding minimizes inconsistencies, or at least make their detection much easier than if the fragments were unrelated.

Besides scaffolding elements, a perspective-centric system can provide a number of features which contribute to prevent inconsistencies. A few example features are listed below. All

features leverage the social aspects described in the previous section. All examples assume that fragment A exists and that user X considers the introduction of a conflicting fragment B.

- All features *maximizing awareness* (c.f. section 5.2.3) of user X that fragment A exists or has just been introduced minimize the risk of divergence, i.e. could prevent the introduction of B
- If user X decides to introduce B, he should be encouraged (or even forced) to use *social search* features (taxonomy, tag-cloud, full-text, ...) in case he wasn't aware of A's existence
- If user X adds fragment B through an interactive user interface, this interface can show *suggestions of similar fragments*, sorted by social relevance. Ideally suggestions should be displayed in real-time, for example in a side-bar. Otherwise suggestions can be displayed as a step prior to committing fragment B.

There are a number of limiting factors weakening the mechanisms above. We can cite operational pressure to immediately introduce item B, inciting user X to ignore the prevention mechanisms. Insufficient sharing is likely to be another problem, if the owner of A is shy or underestimates the maturity or general interest of his fragment and thus does not share it with a sufficiently wide audience, i.e. excluding user X. Language and terminology differences are other well-known issues which may defeat the mechanisms above.

Besides attempting to prevent divergence before-the-fact, it is therefore mandatory to assist in detecting, assessing and resolving inconsistencies after-the-fact.

Detecting Inconsistencies "A Posteriori"

Regardless of the efficiency of prevention mechanisms and the goodwill and discipline of contributors, inconsistencies will occur and must be detected. Detection can be both automatic and manual.

Automatic detection is performed by the infrastructure without human assistance. Consistency rules can be checked and pattern-matching and data mining techniques can be applied to look for potential inconsistencies or duplicates. In addition to comparing model fragments, it can compare instances which help to detect similarities, as illustrated in the example below.

EXAMPLE
As a simple academic use-case, we can consider a central Conference class, with two groups extending it in their own perspectives. The first group adds an attribute ‘deadline’, representing the deadline for paper submissions. The second group adds the same attribute but calls it ‘paper date’, plus a second one called ‘abstract date’. Looking only at the model, the similarity between the two extensions is limited to two aspects: they extend the same class, with at least an attribute of type ‘DATE’. This is not sufficient to raise a similarity flag. However, looking at the instance values the system can detect that when both groups set attribute values for the same conference, ‘deadline’ and ‘paper date’ are always identical, providing a strong indication that the extensions could be semantically equivalent.

Manual detection is performed by people. All users should be allowed to report potential inconsistencies to the system when they see them, whether accidentally or because they were actively looking. In a similar way to the "report spam" button of online email systems, a "report inconsistency" button could provide a way to crowd-source the governance of the global model.

Once detected, inconsistencies must be tracked, assessed and resolved, as presented in the next sections.

Tracking and Assessing Inconsistencies

We consider a detected or reported potential inconsistency an opportunity for convergence, and thus introduce Opportunity as a first class citizen in our meta-model. Since fragment owners will invariably be consulted in the opportunity assessment process, their opinion appears as well as illustrated in the diagram below.

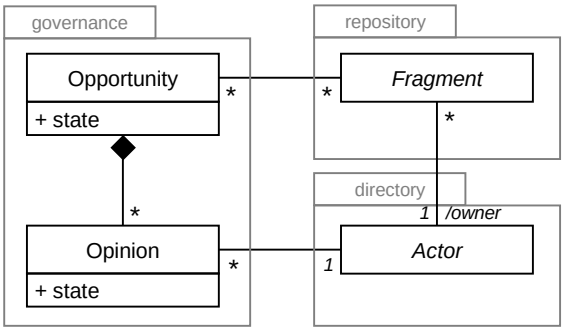


Figure 5-9. New Opportunity and Opinion classes for governance

When a convergence opportunity is detected, manually or automatically, the system must first ensure that it doesn't already exists. If the opportunity is new, an instance of Opportunity is created in state *to-be-assessed*, with the appropriate references to all fragments involved, and the owners of the fragments in question are notified which begins the assessment phase.

The owners can either *agree* that the fragments are similar and that convergence should be discussed, or *disagree*, meaning they think there is a difference which justifies the existence of both fragments. This judgment is represented by an Opinion object, the state of which drives the state of the Opportunity object as illustrated by the diagram below.

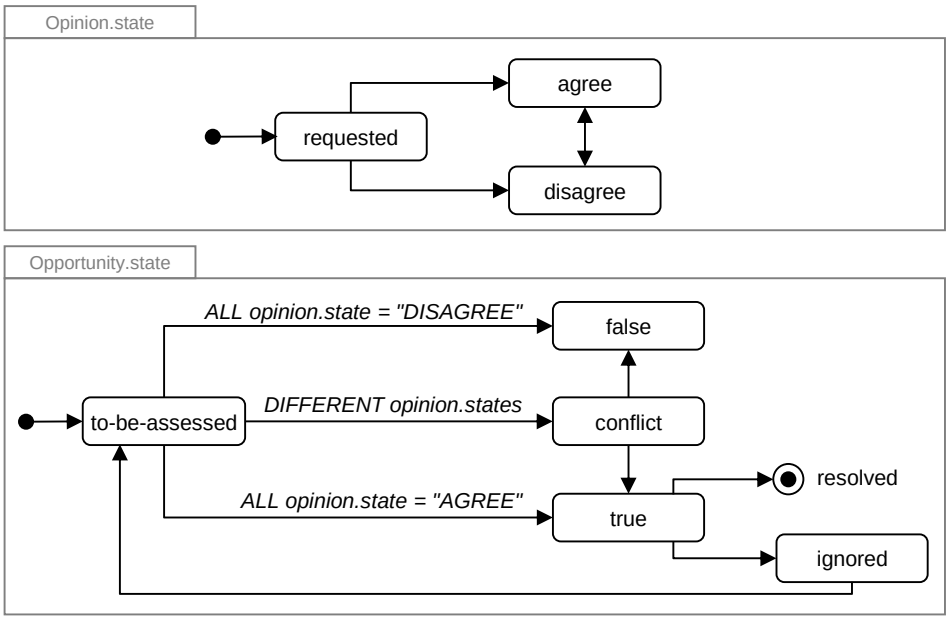


Figure 5-10. Opinion and Opportunity state diagrams.

If all contributors mark their Opinion as *disagree*, we can consider the Opportunity was a false positive. It can be marked as such via the state *false*, but should not be deleted in order to avoid further detection and notification of the same opportunity.

If opinions differ, the Opportunity is marked as *conflict*. Either the divergence is acceptable and remains, or it is not and can only be solved through conflict-management at the human-level without additional support from the architecture.

If all contributors mark their Opinion as *agree*, the Opportunity is *true* and the resolution phase can be attempted.

Resolving Inconsistencies

When all owners involved in a convergence opportunity agree that there is an inconsistency, they can decide *resolve* the inconsistency or to *ignore* it. The decision is based on risk: if the cost of resolving the inconsistency outweighs the risk of ignoring it, it is not worth fixing in some situations [139]. In this case, it is important to re-assess the risk, either periodically or when one of the involved fragments changes, in order to avoid problems like the Ariane 5 maiden-flight explosion [143].

Resolving an inconsistency means *merging* fragments, which implies both the evolution of the schema and the migration of data. Appendix C provides a little more details on this merge operation.

5.3.4. Summary

We have discussed monitoring, community management and inconsistency management, showing that the highly collaborative and distributed nature of social information systems is not necessarily an obstacle to governance but may be an advantage.

Although the deniability of shadow applications is lost, we consider a better visibility into the state and evolution of information systems a major improvement over the present situation.

5.4. Conclusion

Based upon the observation of present information systems, we have presented a fundamentally different enterprise architecture paradigm with the goal of meeting all requirements for an agile information system.

First, we have proposed an alternative decomposition of business applications into perspectives, and of their elements into smaller fragments, providing a more subjective representation of the corporate reality which thus defuses the requirements paradox. This decomposition allows to re-compose profile-specific applications, tailored for each user including completely private data, and to distribute the ownership to the most knowledgeable actors, eliminating the knowledge/influence paradox and the bottleneck of requirements negotiation.

Second, we have described our vision of a social information system, where all actors contribute and share business application fragments, applying social technologies to organize, rank and propagate great numbers of emergent fragments, which can gradually be "promoted" to more central perspectives, enabling the collaborative design and social evolution of the corporate information system.

Finally, we have shown that this shift of responsibilities to a wider community could assist in information system governance, representing a significant improvement over the present state of ungoverned shadow applications.

The next chapter discusses these principles in further detail by describing a possible architecture for a social information system.

6. Architecture

Many different interpretations and implementations of the principles described in the previous chapter are possible. This chapter presents a possible perspective-centric architecture, in order to illustrate the high-level principles and their implications. We will describe this architecture in the form of a set of components, listed below.

- The **foundation components** are the *directory*, managing users and their groups, and *repositories* which host perspectives and fragments
- The **end-user runtime components** supporting our claim for self-tailoring applications are the *weaver* which composes a model driven by the profile of the current user, and the *browser* which interprets this model to construct a user interface, for both regular use and model updates
- The **social collaboration** through sharing, annotating and organizing fragments is supported by the *registry* component, as well as fragment awareness
- Finally, **governance** will be briefly discussed via the *monitoring* component

Each section below presents a specific component of the architecture, gradually building the full picture.

6.1. Foundation Components

6.1.1. Organizational Complexity: the Directory

Present information system architectures typically include an enterprise directory service, usually a set of servers implementing the LDAP protocol [144]. Such a service is used by applications and services alike to authenticate users, and serve as a referential for group membership and other profile data. It is a sensitive component in terms of security, with open read access but write access restricted to administrators. It is optimized for mostly-read access.

A perspective-centric architecture revolves around a similar service. The Directory component is the referential for the social constructs along all dimensions presented in Figure 2-3. The central concept is *actor*. The figure below shows the logical model of the resources under the responsibility of the directory component.

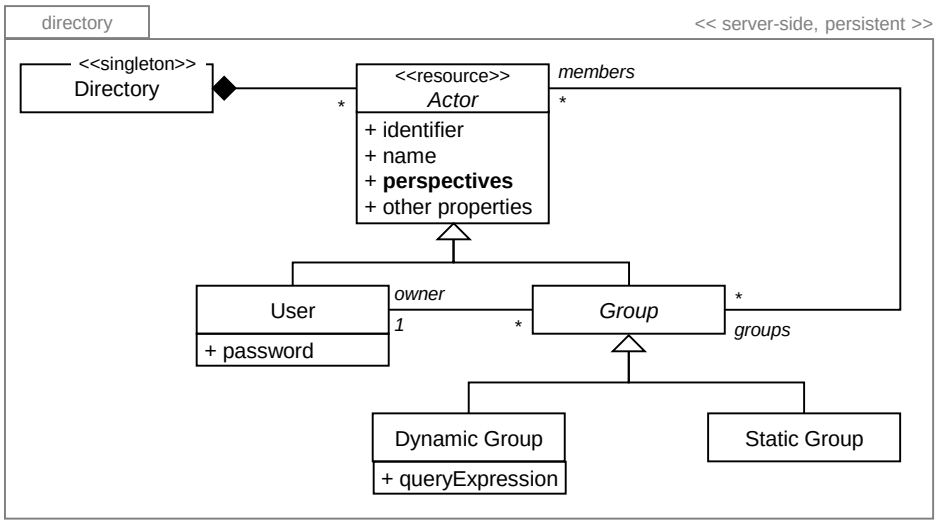


Figure 6-1. Logical model of Directory component

We make the simplifying assumption that a group is owned by a single individual. Groups can be either static or dynamic. Static groups have an explicit list of members. Dynamic groups have an expression which determines the set of members. The <<singleton>> annotation indicates that there is typically only one logical instance of Directory, although it is common to have several physical instances for load-balancing and high-availability reasons.

The instance diagram below illustrates typical usage of the directory component.

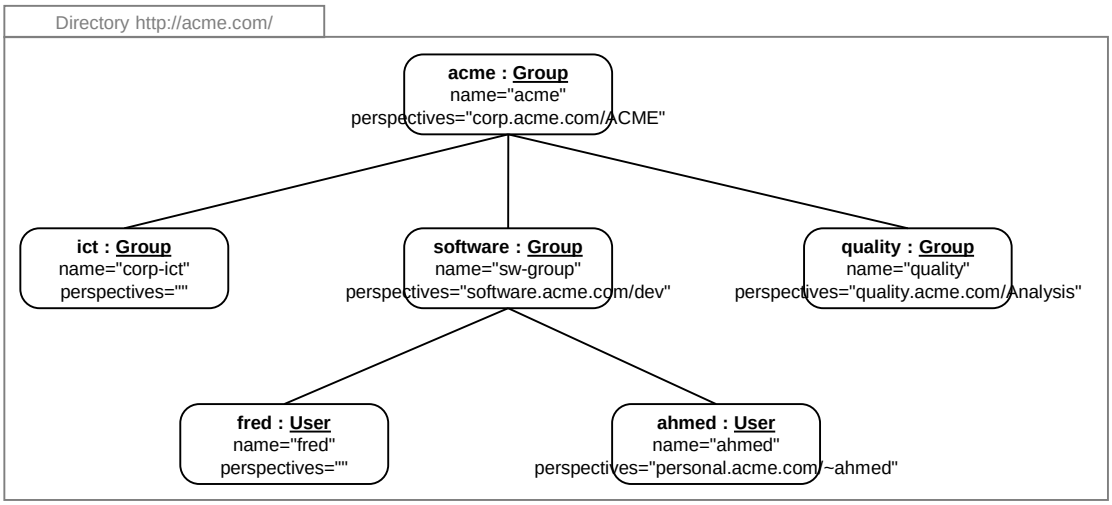


Figure 6-2. Example Directory instances.

The main difference w.r.t. present directories are that all actors are first-level citizens, and that they need to be annotated with the (possibly empty) list of their perspectives. This is perfectly achievable with present enterprise directory technologies.

The main usage of the directory is at initialization-time, where it can provide the complete profile of the connected user, i.e. his groups and perspectives, as illustrated below by an example reply to an authentication request.

```
<User id="123" name="fred"
  <perspective url="http://personal.acme.com/~fred"/>
  <is-member-of>
    <Group id="456" name="sw-group">
        <perspective url="http://software.acme.com/dev"/>
      ...
</User>
```

Figure 6-3. Example reply of the Directory component

Due to the aforementioned security constraints, some administration operations must remain centralized, mainly the creation of a new user. However, in a perspective-centric architecture it is important that any actor can create new groups, and can attach perspectives to the groups he owns.

The figure below shows the component diagram with only the Directory component and its public interface allowing to create, retrieve, update and delete actors.

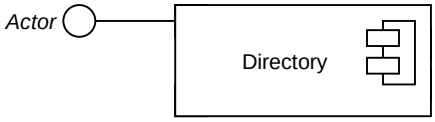


Figure 6-4. Component diagram: Directory

6.1.2. Application Fragmentation: Repositories

The Repository service hosts the bulk of business applications, i.e. the fragments which can be composed to form applications, organized around perspectives. The class diagram below shows the logical model of the resources under the responsibility of a repository component.

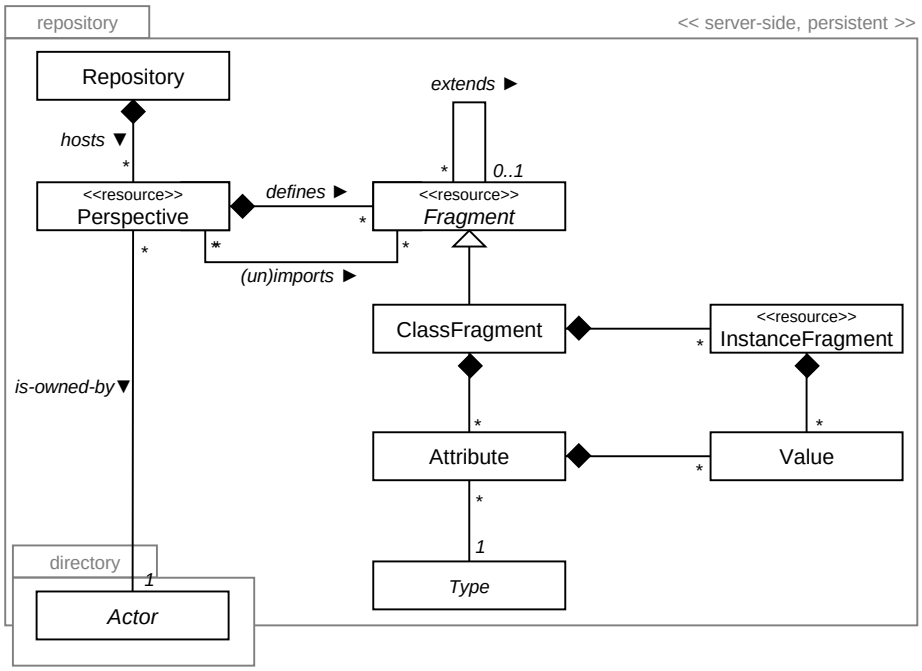


Figure 6-5. Logical model of the Repository component

The example instance diagram below shows the perspective-centric equivalent of a simple application which manages "Requests". This translates into a Perspective "luxury", which

defines a single ClassFragment "Request" with two attributes "title" and "state". The owner of this perspective is actor "corporate-IT", thus representing a typical official application. The diagram provides 3 example instances of requests with the associated attribute values.

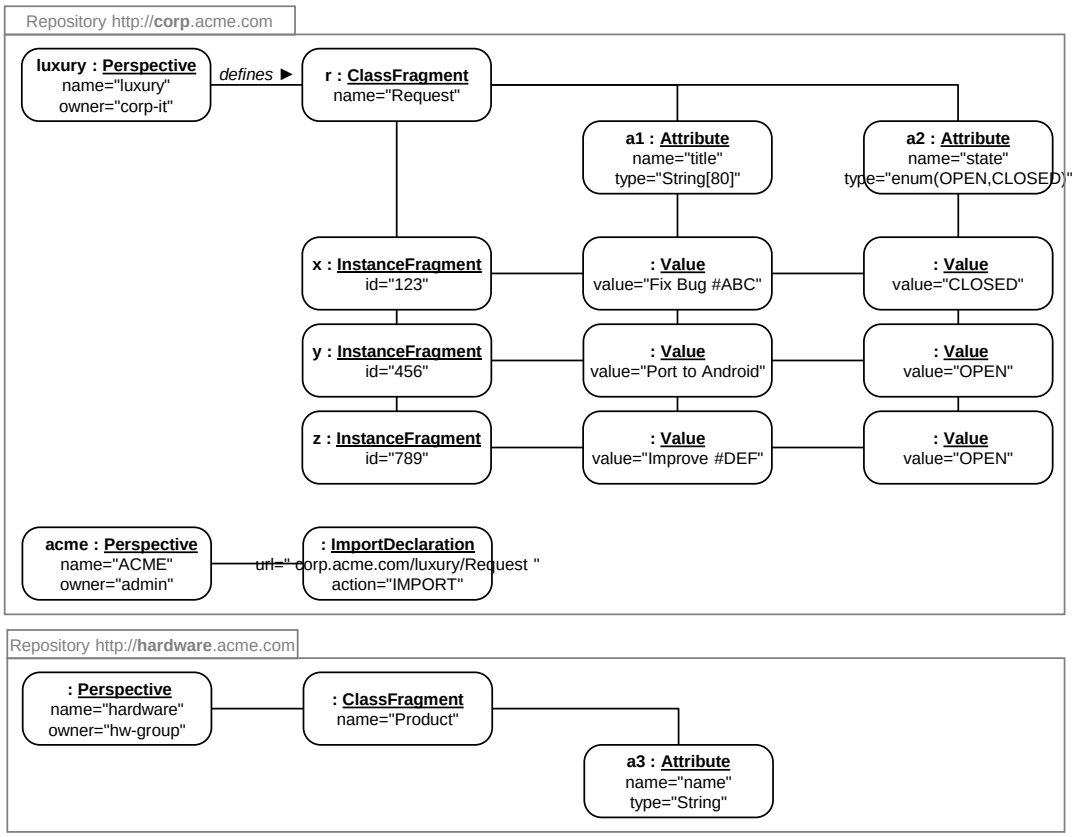


Figure 6-6. Example instance diagram of two Repositories hosting three Perspectives

Propagating the "Request" object to the entire company is done by importing the ClassFragment in the ACME perspective, associated with the top-level group in the corporation's organization as shown in Figure 2-3. All other departments thus inherit ClassFragment "Request", through a mechanism we will describe in detail in the next section.

Besides the official application, the "hardware" perspective defines a simple class "Product". In an application-centric architecture, this would most likely have been a separate application, owned by the software department.

The examples above illustrate the fact that perspectives serve two purposes.

- *installation*: the "luxury" perspective is similar to a third-party application, completely self-sufficient, managed by a corporate IT department and intended for use by the entire corporation.
- *deployment and configuration* : by importing the Request ClassFragment, the ACME perspective makes it available to all its members, in this case the entire corporation. This level of indirection separates the deployment and configuration from the installation, making it easy to share an installation between multiple groups with divergent configuration requirements.

It is worth noting that perspectives can be hosted on different repositories and thus different physical servers. Even when hosted in the same repository, perspectives are isolated from each other. As a consequence, cross-perspective references are expressed with URLs.

The next instance diagram shows the repository of the "quality" group, which hosts their "Analysis" perspective on yet another server (http://quality.acme.com/). It extends the concept “Request” from the “luxury” perspective with a new attribute "reason for delay". Two instances of request have values associated with them.

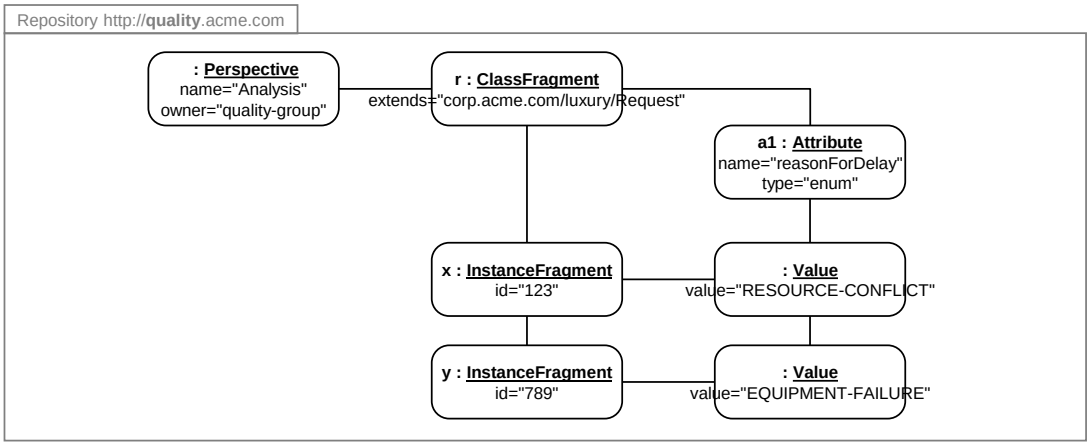


Figure 6-7. Example instance diagram of a Perspective defining an extension

Though a simplification of our running example, this shows how shadow applications can be avoided. Even in a situation where an extension is really group-specific and of interest to nobody else, or where for any other reason there is no hope to influence the official application, or where the quality group absolutely wants to keep the ownership of the extension, there is no need to introduce a completely new application. Instead, the "Analysis" perspective hosts only the *delta* with respect to the "official application", which gets overlaid by the composition mechanism described in the next section.

The third instance diagram shows the individual perspective of employee "Ahmed". He introduces yet another extension of the "Request" concept, adding attribute "risk". Also, as a member of the software group, he inherits the "product" class. But since he is not interested in products, he chooses to remove it from his environment through the "unimport" declaration.

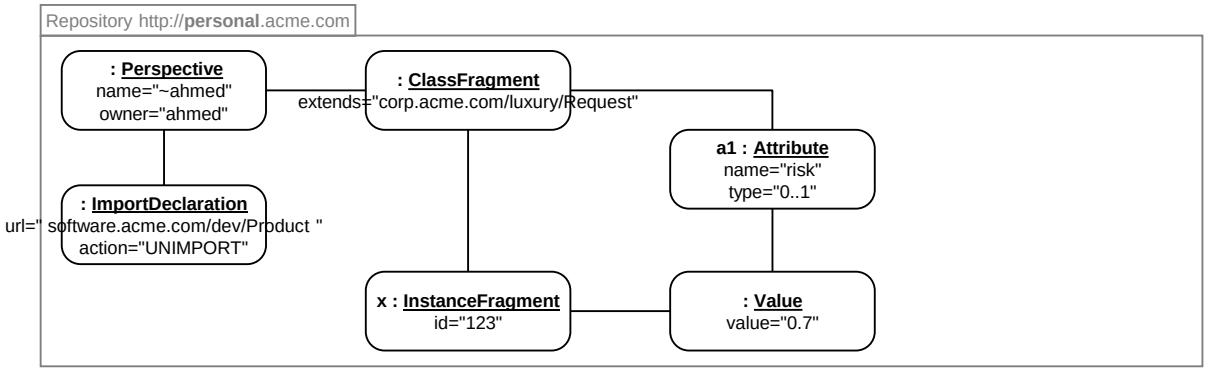


Figure 6-8. Example instance diagram of a private Perspective

The class diagram in Figure 6-5 has pictured "Type" as abstract. The class diagram below shows examples of derived concrete types.

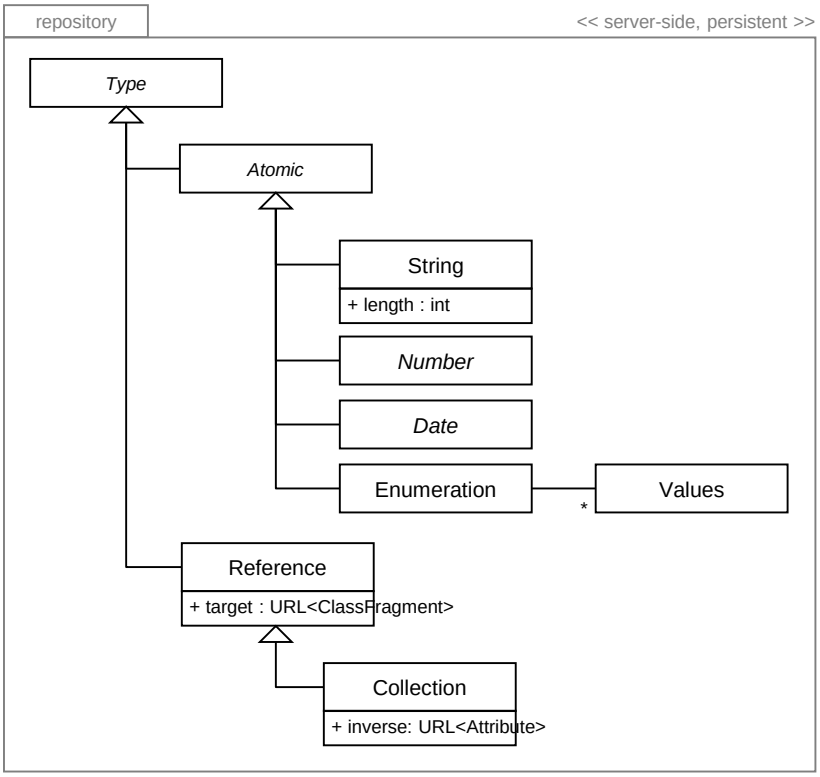


Figure 6-9. Minimum set of concrete type classes

The "Reference" type allows to represent associations between business concepts. Since the target of the reference is a URL, it allows to connect instances across servers, as illustrated by the next and last instance diagram. This perspective shows that individual "barney" needs to track two additional aspects of "Request" objects: which "Product" they relate to, and what their specification is. Even though "Request", "Product" and "Document" are hosted on different repositories, and this extension lives on a fourth one, these declarations demonstrate how perspectives provide an integration mechanism when actors need to tie together previously unrelated entities.

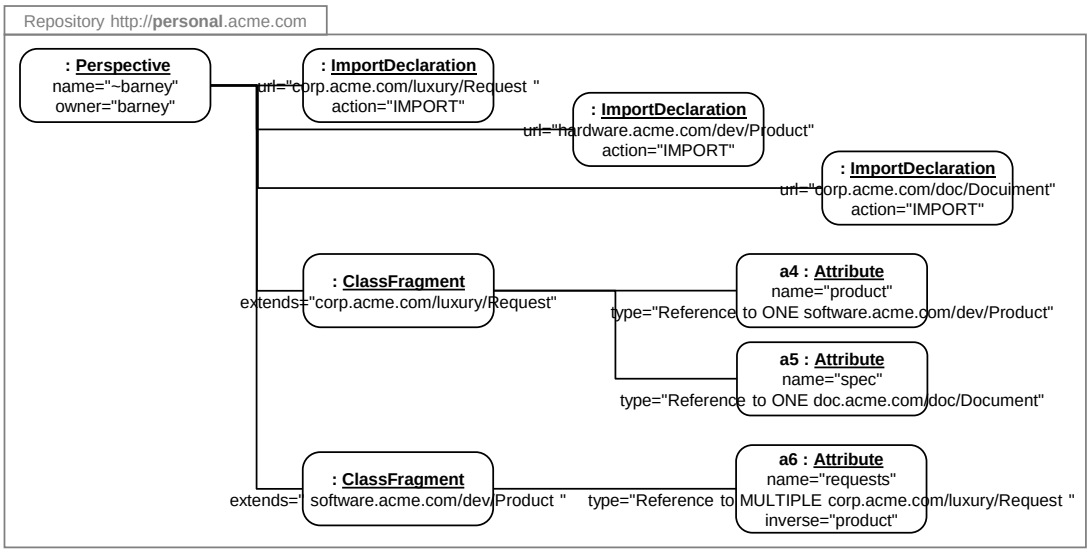


Figure 6-10. Example instance diagram of a Perspective connecting unrelated Fragments

The Repository meta-model in Figure 6-5 provides a single type of fragment granularity, i.e. the ClassFragment. In order to build more complex business applications, composite Fragments are necessary to manage related fragments together as a consistent whole. We can envision a generic PackageFragment construct as illustrated in the figure below.

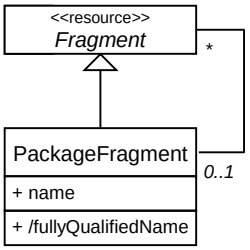


Figure 6-11. A necessary evolution of the meta-model: PackageFragments

However simple it may seem, the implications of such a composition mechanism are beyond the scope of this document. Finer levels of fragments would be interesting as well: Attributes and Types could be reused across various ClassFragments. Again, this appears like a simple change of multiplicity in the meta-model but, though perfectly possible, introduces a complexity beyond our present discussion.

The independent nature of fragments constitutes an oversimplification as well. We have presented two kinds of dependencies among fragments: extension and reference. It is possible to envision other relationships, like the “requires" dependency in component-based software engineering [145].

When looking at the meta-model of Figure 6-5 and the instance diagrams in this section, the similarities between a Repository component and a regular database server are apparent. Indeed, a Repository component only represents a fairly thin layer on top of a database server. Additionally, we think perspectives could provide a natural mechanism for both vertical and horizontal data partitioning [146].

The figure below shows the component diagram at this point, with the unchanged Directory component, and the Repository component requiring Actor management and providing Perspectives and Fragments as its public interface.

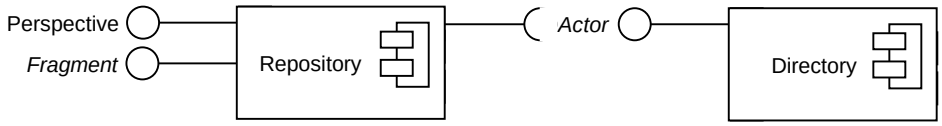


Figure 6-12. Foundation Components: Repository and Directory

The public interface of the Repository component is voluntarily simple, in order to enable legacy system integration by writing wrapper components which expose the legacy systems' model and instances as Fragments. Section 7.3 presents a prototype implementation of such a wrapper.

A Directory and several Repository components provide the server-side foundation for a perspective-centric architecture. The repositories host isolated perspectives and fragments, and the Directory provides the map which indicates how these must be composed for a given actor. This composition is the responsibility of the Weaver component, described in the next section.

6.2. End-User Runtime Components

6.2.1. Profile-Driven Fragment Composition: the Weaver

The purpose of the Weaver component is to provide a given Actor with his own subjective view of the information system. It constructs a unified model, weaving together the relevant fragments and extensions to form a consistent set of *elements*, effectively building the *model* underlying the current actor's profile-specific application. The class diagram below presents the public interface of the composition mechanism, which also shows that the model has a lifespan limited to the duration of a session. As a consequence, a profile-specific application only exists while a user is connected. Between sessions, the composition does not exist.

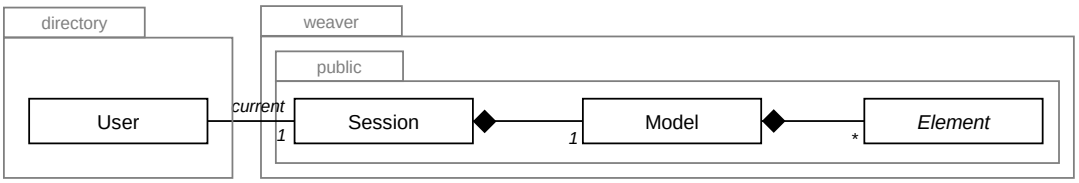


Figure 6-13. Simplified public interface of composition mechanism

The above interface hides the fragmentation of application elements from higher-level components, giving the illusion of a set of atomic elements. The diagram below shows the internal concepts[20] the Weaver must leverage to instantiate elements.

[20] In the context of the Weaver component, the internal classes should be considered *proxy* classes [139], i.e. each instance is a local representation of one or several remote objects.

In order for a model to reflect the subjective view of the *current* actor, a graph of actors is built by traversing the groups which he is a member of. An example of such a graph is shown in Figure 6-16 below. The model thus corresponds to a set of Perspectives, inferred from the graph of Actors. An Element has one root Fragment and potentially multiple extension Fragments from different Perspectives.

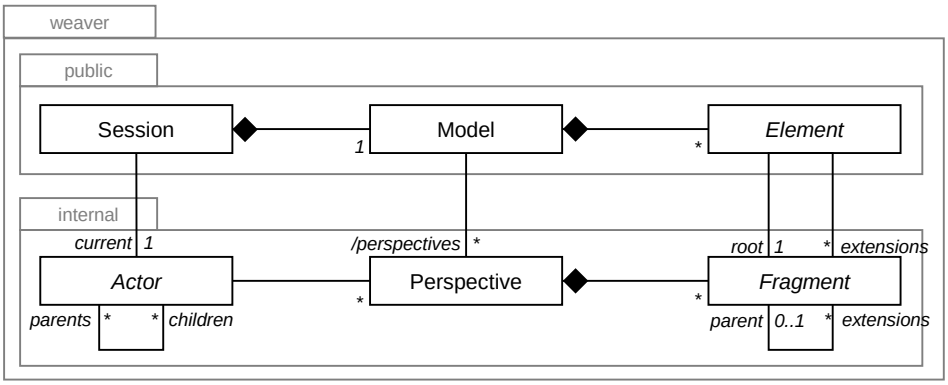


Figure 6-14. Classes underlying the simplified public interface

To illustrate the principles driving the composition logic, we will use an example diagram representing the actor graph and associated perspectives, using the following notation.

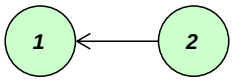
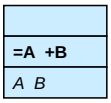
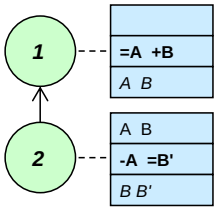
Symbol	Meaning
	1 is an *Actor*, either User or Group
	Actor 2 is *member* of Group 1
	A Perspective. The upper part represents *inherited* fragments, the middle part *declarations*, and the bottom part the resulting set of fragments
	Perspective *defining* fragment A (symbol "=") and *importing* fragment B (symbol "+") in the middle part. The bottom part thus shows available fragments A and B.
	Actor 2 *inherits* fragments A and B through his membership of Group 1 in the upper part. In the middle part, actor 2 *unimports* fragment A (symbol "-") and defines fragment B', *extension* of B (the prime symbol denotes extension).

Figure 6-15. Explanation of notation elements for the next diagram

We will start with the simplifying assumption that an Actor has 0 or 1 Perspective. We will thus call "Perspective 0" the Perspective associated with group 0.

Figure 6-16 below shows a user 8 belonging to multiple groups with various declarations, presenting a number of possible combinations of primitive operations. It illustrates the

propagation of fragments and the resulting set of elements when weaving all fragments together for each actor. Only the paths which user 8 is a member of are represented.

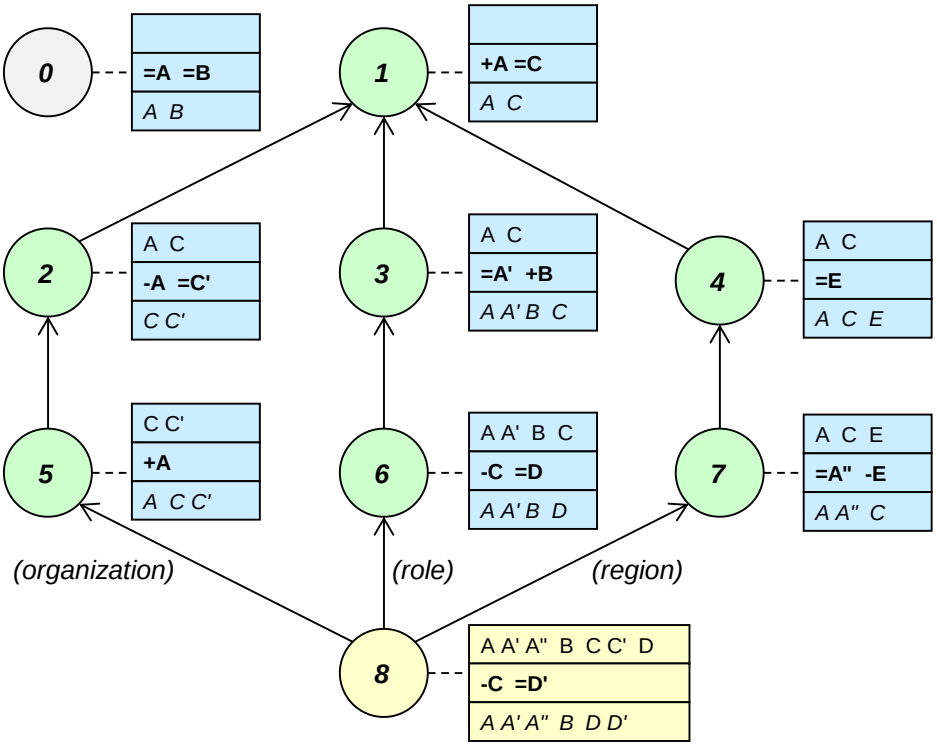


Figure 6-16. Example Actor-Perspective graph, with associated declarations

- User 8 is a member of groups 5, 6, and 7. All these groups are member of the same corporation, and represent the users' organization, role, and region respectively.
- Perspective 0 defines 2 fragments, A and B. This situation represents for example an installed third-party tool with two modules.
- Group 1 is the root group, representing the entire corporation. Its purpose is to select the fragments which are available to all employees. In the example it imports A from Perspective 0, and defines an additional element C.
- Group 2 is member of group 1, for example a business-unit. Group 2 thus inherits fragments A and C. However, it decides that element A is not relevant for its activity and does not want to push it to its members, and thus unimports it. It defines an extension of C called C'. Its member group 5 (a department) disagrees with the parent group and imports A. Through path (organization), Actor 8 thus inherits A and C and C'. Likewise, through the (role) and (organization) paths, different fragments are defined, imported and unimported.
- There is an apparent conflict between paths (organization, region) and path (role). Both (organization) and (region) consider fragment C as relevant, but (role) does not. By default, unimport is considered a lower priority operation than import and define, for the reason that if C is relevant for at least one community 8 is a member of, it is potentially relevant for 8. In the example this is not the case however, because user 8 in turn decides to unimport C.

- The diagram shows that unimporting C in perspective 8 also disables the inherited extension C', thus illustrating our assumption that extensions don't make sense without the associated root fragment.
- User 8 also defines a private extension D', and thus the final elements to be assembled are element A (fragments A, A' and A"), element B, and element D (fragments D and D').

Building the actor-perspective graph in a robust manner is not trivial. Requirement R6 (resilience) states that even when fragments are missing, the user must be able to interact with the remainder of the model. This implies asynchronous communication between the weaver component and the various repositories. Appendix D provides a detailed description of this recursive asynchronous model composition.

In order to propagate the resilience characteristic to its client components, the Weaver must expose an event-driven behavior. The public interface presented in Figure 6-13 must thus be completed with event classes as illustrated in the diagram below.

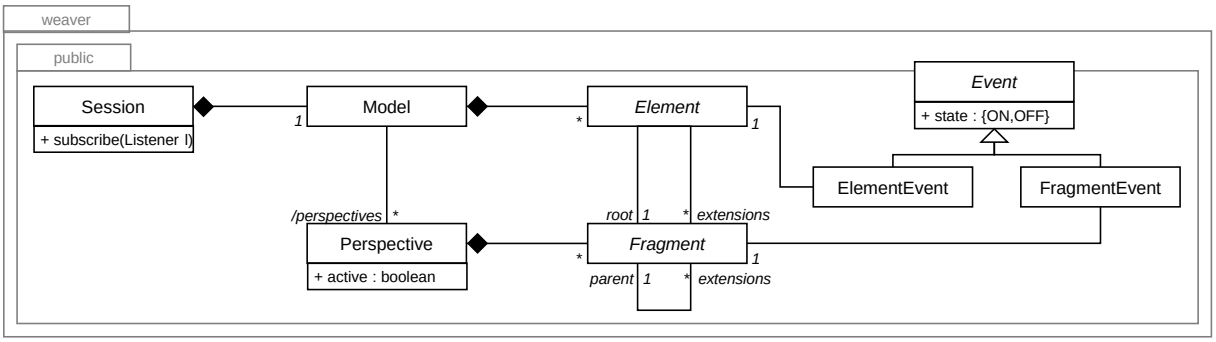


Figure 6-17. Complete public interface of the Weaver component

A surprising twist of this asynchronous behavior is that it is possible to receive a fragment, notify the session accordingly, and receive at a later point in time the description of a perspective with higher priority which unimports the fragment in question. It is thus necessary to indicate in the event whether it enables or disables the related fragment or element.

It is interesting to provide client components with the capability to enable and disable perspectives, for example for users with many roles to filter elements, or for confidentiality reasons when sharing a screen with another person. Another common situation could be a remote support interaction, where it can be expected that application support people will request to disable extensions in order to understand the users' question. User-controlled enabling and disabling can leverage the same events.

In data-centric applications, the concepts of element and fragment must be specialized, as illustrated by the diagram below which also introduces the notion of *instance* in its perspective-centric sense, compliant with the usual definition, i.e. a set of *values*.

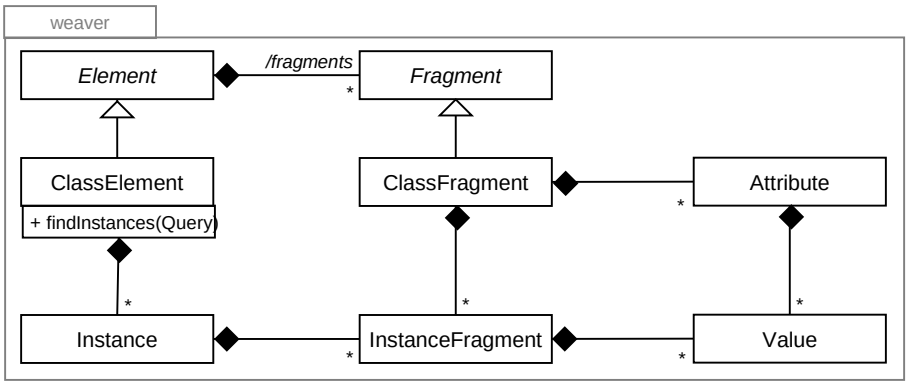


Figure 6-18. Instances and InstanceFragments

An instance is the composition of InstanceFragments from various repositories, which must satisfy the R6 (resilience) constraint. The model composition mechanism described in appendix D must thus be generalized to compose instances, including the notification of InstanceEvents.

6.2.2. Model-Driven User Interface Construction: the Browser

An application user interacts with the information system through *forms*. This section describes the user interface construction mechanism introduced in Figure 5-7, represented by the Browser component which constructs a presentation layer on top of the dynamic, asynchronously composed model presented in the previous section, both for *instance manipulation* (regular application usage) and for *model manipulation* (adapting the application, i.e. end-user modeling).

Instance manipulation

The browser provides a standard pattern of forms for instance manipulation, allowing all common end-user manipulations (CRUD). The diagram below shows the flow between the 5 main forms of this standard pattern, representing the manipulation of instances of a given ClassElement x. A more detailed description of each form is provided in appendix E.1.

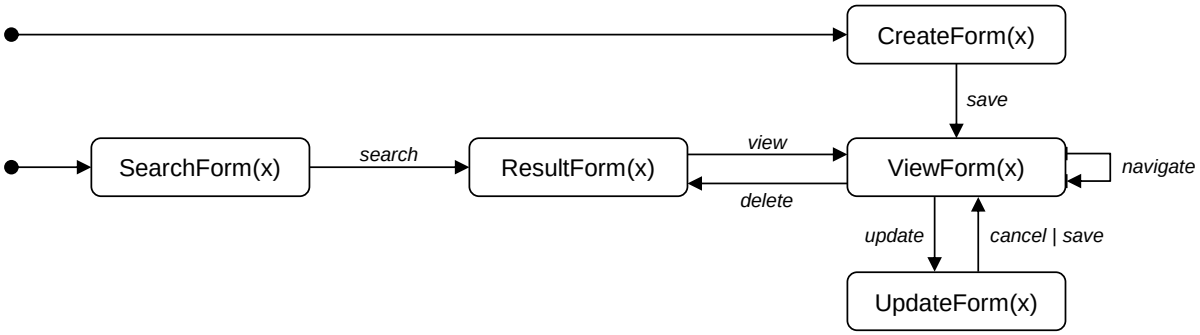


Figure 6-19. Standard form pattern for instance manipulation for a ClassElement x

The content of each form depends on the ClassElement it is associated with. The figure below shows a simplified instance diagram of ClassElement “Request”. The full instance diagram for this example is provided in appendix E.2.

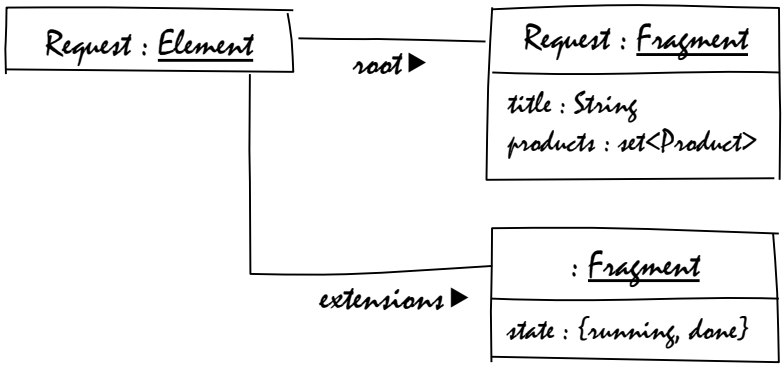


Figure 6-20. Simplified ClassElement "Request"

When instantiating a form for a given ClassElement, the browser iterates over all attributes of all ClassFragments, and creates the widgets according to the nature of the form (search, update, ...) and the type of the attribute. The example below shows the user interface which is constructed when applying the standard pattern (Figure 6-19) to our example ClassElement (Figure 6-20), including the navigation among these forms. Example classes underlying such a construction mechanism are presented in appendix E.3.

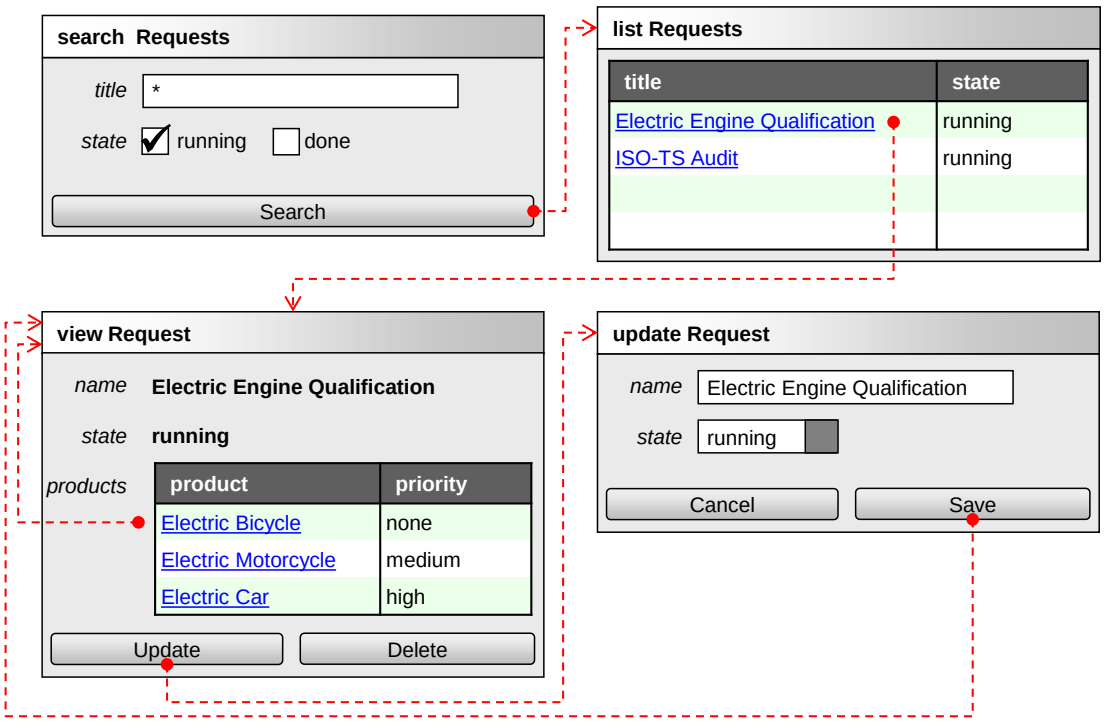


Figure 6-21. Result of the user interface construction mechanism on the example ClassElement

Besides class-specific forms, at the top level the user interface must provide a menu proposing to both search for and create any kind of ClassElement, as will be illustrated in the next chapter presenting our prototype implementation.

Beyond the standard interaction model allowing to manipulate *instances*, the browser must provide a way for an end-user to interact with the underlying *model*, as described in the next section.

Model manipulation

We have mentioned that every employee becomes a potential provider of fragments, private or shared with other actors. Usability is a key enabler for this, and we consider the success of spreadsheets as a good indicator that employees will contribute pieces to the information system when provided with the right tools.

Spreadsheets do not really distinguish between design and usage or between model and schema, thus providing a form of design by example, naturally geared towards experimentation. They are a perfect example of the gentle slope principle, where even novices can create tables by using simple and universal conventions (the first row holds the columns names); more advanced users can indicate the type of data, create reference tables and simple joins via lookup functions, and describe simple calculations ("formulas"); experts have a rich set of features blending declarative and imperative programming styles.

Given the foundation concepts of perspectives and fragments, it is possible to envision the same set of features for end-user business application extension. As stated in the example for R3 (ease-of-modification), it is possible to allow users to click next to the last column of a "Request" table and let them type a column header "difficulty", and start filling values. Behind the scenes this can create a fragment extending "Request", with a "difficulty" attribute of either default type string or a type inferred from the values. Right-click contextual menus or other more intuitive mechanisms can provide the complex operations required for advanced users and experts.

The Browser component must thus provide both an automatic user interface construction mechanism and an intuitive end-user modeling and development environment. Besides just adding attributes, this environment must allow other operations, like importing available fragments, unimporting irrelevant inherited ones, or create completely new fragments.

The user interface design effort this implies is beyond the scope of our study, but a minimal, less ambitious forms-based implementation of these end-user development features has been implemented in our prototype and will be illustrated in chapter 7.

The diagram below shows the set of components introduced so far.

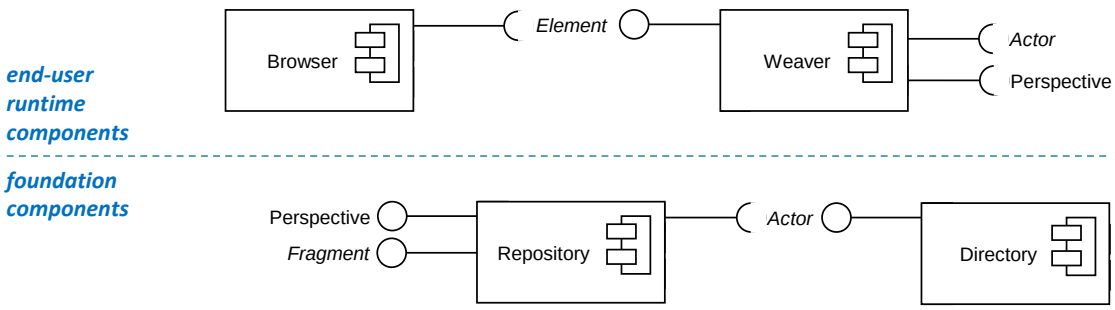


Figure 6-22. Component diagram including browser

These components are the rough equivalent of present business applications. Besides being designed for agility, they enable new forms of *collaboration* and *governance* described in the final two sections.

6.3. Social Collaboration

Continuous collaboration is a central aspect of our proposal, for which we have proposed social mechanisms. In our reference architecture this is the responsibility of the registry component. Like the directory component and unlike repositories, the registry component is central and serves as a catalog of available fragments, providing the following operations.

- *share* fragments with other actors, implementing the "export" primitive and thus requirement R12 (sharing)
- *annotate* fragments, providing the necessary information to determine the relevance of available fragments for a given actor as dictated by R14 (relevance)
- *search* for and *notify* about available fragments, implementing R13 (awareness)

6.3.1. Fragment Sharing

The owner of a fragment can share it with other actors, which we have called export. At the minimum, the owner must be able to express the set of target actors, as presented in the diagram below.

Figure 6-23. General case of ExportDeclaration

Beyond the general case, specific subtypes of Fragment can require specialized ExportDeclarations. In the case of ClassFragments, the owner could choose to share only a subset of all instances, a subset of all attributes, and grant various fine-grained permissions at both the instance and attribute level. This is very similar to the selection and projection operations in relational algebra [17] and to present database authorization mechanisms, as illustrated by the class diagram below. It provides an example of a subclass of ExportDeclaration specialized for ClassFragments. It introduces the concepts of View and Permission which we will not further describe in this document but are used here to illustrate a specialization. Since import and export are symmetrical operations, it could be interesting to provide similar specializations at import time.

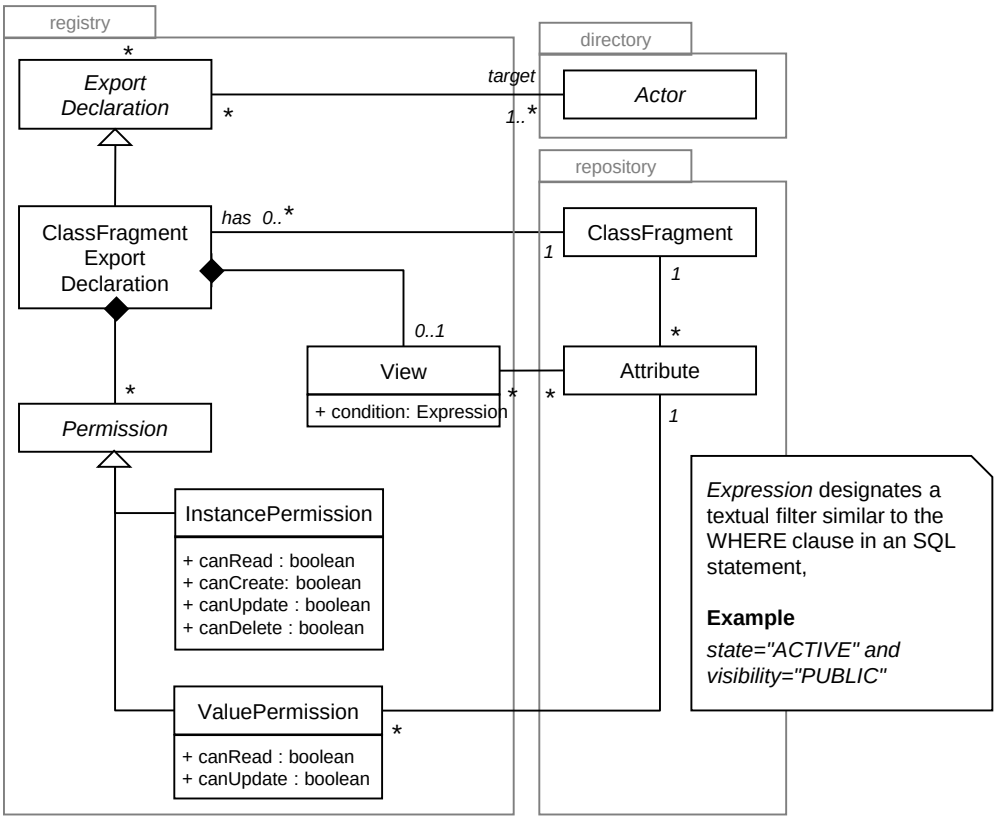


Figure 6-24. ExportDeclaration specialization for ClassFragments

Sharing an extension with the owner of the root fragment is a specific case, which probably deserves a dedicated property in the model. If fragment B is an extension of fragment A, the owner of B can choose to either keep his adaptation hidden from the owner of A or to make the owner of A aware of the adaptation. In the first case, the owner of B puts the emphasis on confidentiality (requirement R4). In the second case, he considers traceability (R10) more important and minimizes the risk of A changing without him being forewarned.

6.3.2. Fragment Annotation

In section 5.2, we have proposed to leverage social mechanisms to organize the huge number of fragments a big corporation can potentially produce. Although it would technically be possible to store social annotations next to fragments, it appears safer to centralize the opinions which annotations express in a component managed by an impartial authority. The diagram below presents the most common social annotations applied to business application fragments.

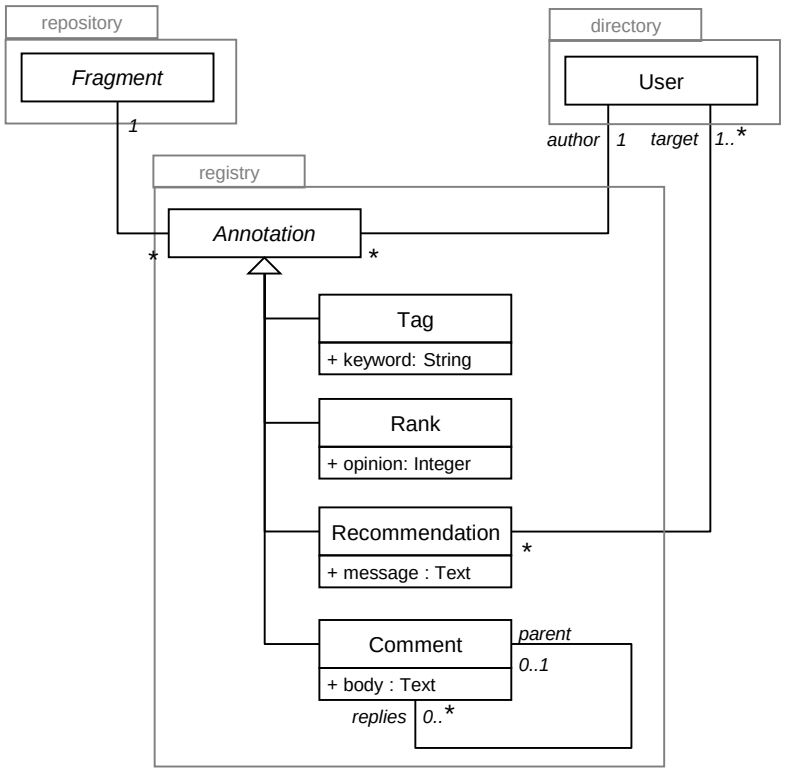


Figure 6-25. Social Annotations for business application fragments

A *Tag* annotation associates a free text keyword to a fragment, in order to enable folksonomies as described in section 5.2.2.

A *Rating* annotation expresses the quantified overall opinion of a given user on a given fragment. In the simplest case, these are the "like" and "don't like" buttons found on many social networking sites. A slightly wider range of values is provided by the also very popular "star rating" system. Both types of rating provide valuable input to recommender systems as discussed in section 5.2.3.

A *Recommendation* annotation represents a direct message to an actor, a suggestion for him to look at a particular fragment and take action.

A *Comment* is part of a discussion thread attached to a given fragment, and can hold questions, answers, opinions, and anything else. It could be interesting to consider Comments as fragments in their own right, in order to organize them with respect to their tags and ratings, as made popular by question-and-answer oriented knowledge sharing sites like stackoverflow[21].

It is important to remember that in an enterprise environment a user is an authenticated employee of the corporation, and as such will exhibit professional behavior when annotating fragments, providing a more beneficial setting for social technologies than an anonymous consumer space.

[21] www.stackoverflow.com

The new generation of workers has grown up with social networks as an important part of their lives, and can thus be expected to spontaneously annotate the business application fragments they work with.

In settings where users do not contribute annotations spontaneously, they can be encouraged to do so. If detected by the system that they use certain elements but have no annotation, a low-frequency popup window can suggest contributing ratings and tags. If further incentives are needed, it could be made a groups' goal to have each member contribute at least N opinions on the information system. Following the recent trend of serious games, [147] proposes a tagging mechanism which simultaneously entertains users and encourages them to contribute annotations.

Annotations can be interpreted directly by users when public, but are most useful when leveraged by the services presented in the next sections.

6.3.3. Fragment Search

When facing a new requirement, a user must be able to first search for existing fragments satisfying his needs before deciding to introduce a new fragment himself.

Traditional full-text search mechanisms can be applied, with decreasing weight for terms found in the fragments' name, short and long descriptions. In addition, full-text search can incorporate tags and text from aforementioned Comments and Recommendations. Full-text search can allow for logical operators and parenthesis. In addition, it is common to provide an "advanced search" mode, providing more fields like the owner, the perspective, the modification date, the root fragment in case of an extension, etc. Advanced queries can be entered via a form or with a text-based syntax for full flexibility.

```
manufacturing
```

```
owner:/acme/ICT/collaborative-tools
```

```
extends:Task AND perspective:*.china.acme.com/*
```

Figure 6-26. Three example full-text queries

A critical aspect of searching through a high number of items is the ability to determine the relevance of the candidate items, and present the results with the highest relevance first. In section 5.2.3, we have proposed to apply recommender algorithms used in the consumer space to application fragments, and have described the available data for computing a profile similarity score and fragment ranking.

Besides search, these relevance scores can be leveraged to proactively notify users, as described in the next section.

6.3.4. Fragment Notification

In order to improve awareness of interesting components, the registry component can provide a mechanism notifying actors of newly available relevant fragments. This can be envisioned as

a configurable search running in the background on behalf of an actor. Besides the fact that the notifications concern application fragments and not instances, this is similar to common business application notification mechanisms and thus does not require more details than provided in the figure below.

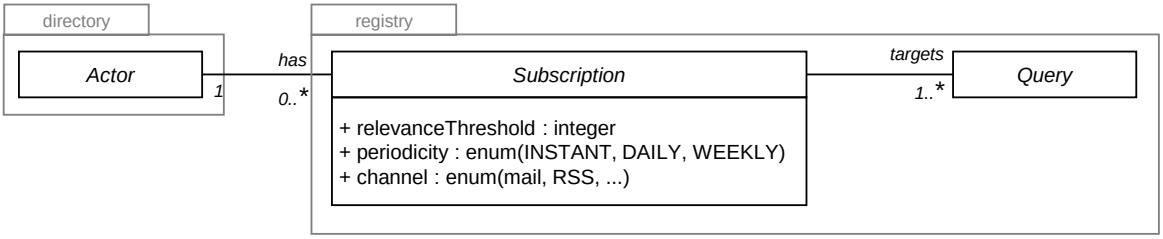


Figure 6-27. Simple notification mechanism

6.3.5. Summary

The registry component provides a number of central services to the browser, enabling the sharing of fragments, their annotation, a central place for searching fragments ordered by relevance, and possibly a notification mechanism to raise awareness.

6.4. Governance of User-Contributed Fragments

Indicator management is a prerequisite for system governance, as presented in section 5.3. We envision a dedicated Monitoring component collecting indicators from all other components in order to observe the evolution of a perspective-centric information system and estimate its soundness.

The main purpose of indicators is to be presented in the form of dashboards, for use by IT management, corporate management, community management and ideally all actors. We think the reuse of present enterprise monitoring tools (often SNMP[22]-based) can help in the governance of a perspective-centric information system. As an example of such tools, the figure below shows two dashboards used to monitor the volumes, load and performance of a set of 250 databases in an industrial production environment.

[22] Simple Network Management Protocol, a family of standards used for monitoring networks and other enterprise resources [172]

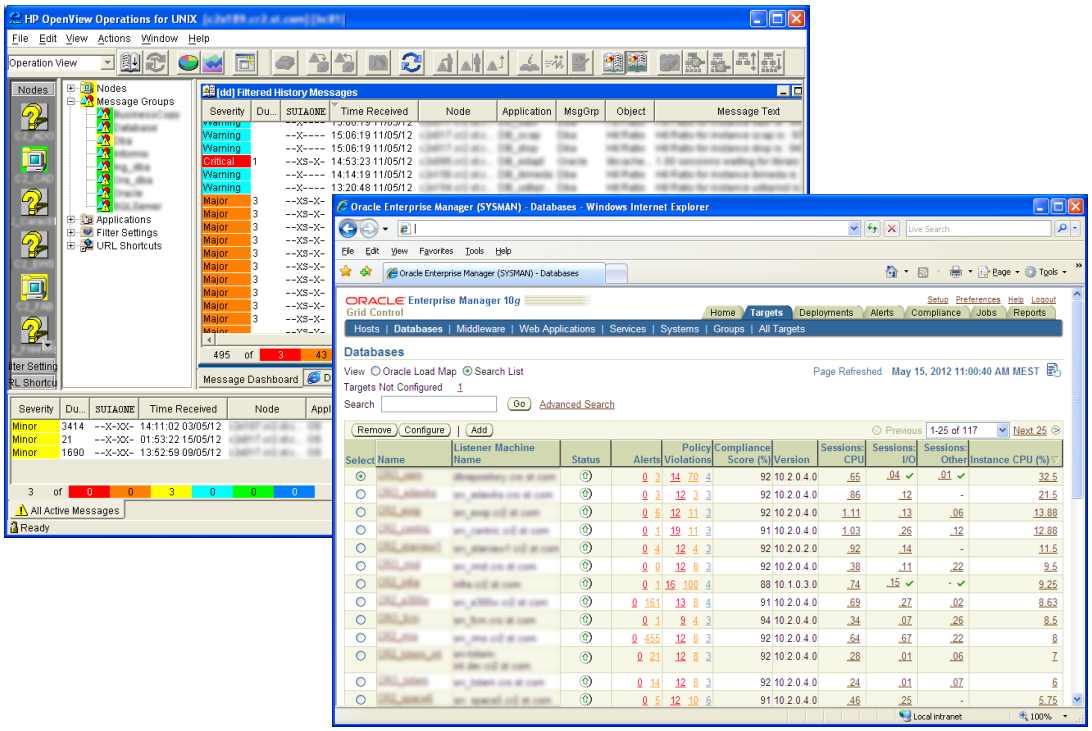


Figure 6-28. Dashboards for database monitoring in an industrial environment

6.5. Conclusion

In the previous sections, we have presented a possible perspective-centric architecture by describing a set of components with their responsibilities and interactions. The figure below shows the all components with their dependencies.

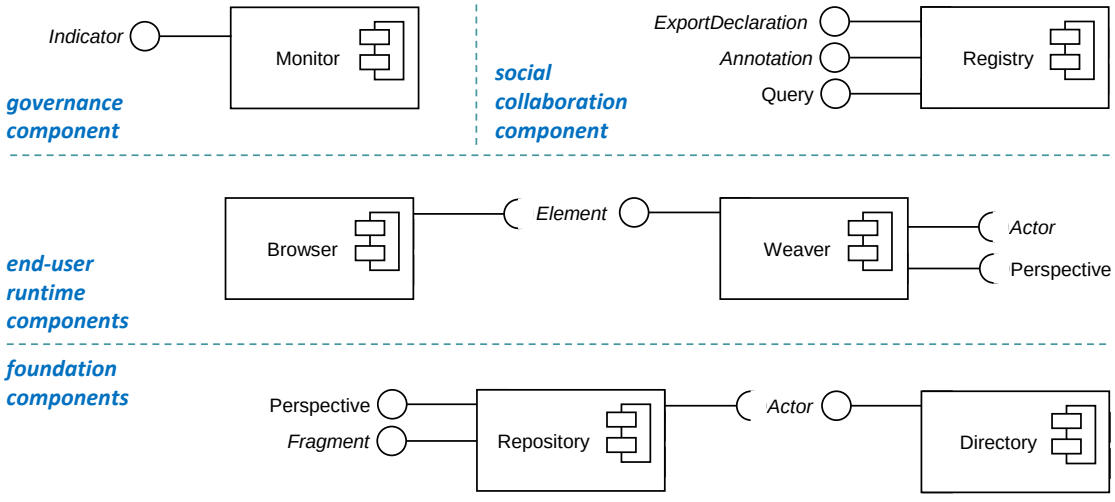


Figure 6-29. Complete set of perspective-centric architecture components

7. Prototype Implementation

We have implemented a first prototype of a perspective-centric application, in order to evaluate the impact of perspectives on the user experience, assess the technical feasibility, identify difficulties and act as a vehicle for experiments.

The following sections present the subset of components we have implemented, an overview of the user experience during instance manipulation and model adaptation, an example of legacy integration, and a first FeatureFragment.

7.1. Overview

In order to have a functional prototype, we have focused on the foundation components, i.e. the directory, the repository, the weaver and the browser. The figure below shows the interactions of these components during the initialization phase (1, 2 and 3) and regular instance manipulation (4).

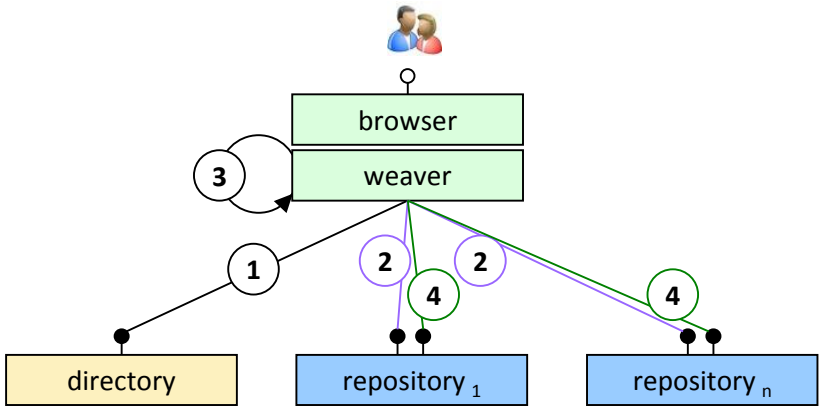


Figure 7-1. Architecture of the prototype and main component interactions

In step (1), the weaver authenticates the user and gets as a reply the full graph of his groups and perspectives. This allows the client to (2) request all perspectives and the associated fragment definitions from the various repositories involved (see figure below). Receiving a fragment triggers the (3) weaving mechanism which composes the associated elements. Regular use is then equivalent to any distributed system, where accessing an object translates into multiple requests (4).

The communication between components uses the REST architectural style [148] over the HTTP protocol. The figure below shows examples of the main requests in Figure 7-1 and XML

snippets of the corresponding replies (when applicable, identifiers have been replaced by labels for readability purposes).

1 ***http://directory.acme.com/Actor/barney***

```
<Account uid="barney">
  <owns>
    <perspective url="http://slow.acme.com/~barney"/>
  </owns>
  <is-member-of>
    <group name="quality">
      <owns>
        <perspective url="http://fast.acme.com/Quality"/>
       ...
</Account>
```

2 ***http://fast.acme.com/Quality***

```
<Perspective name="Quality" owner="...">
  <defines>
     <Class name="Delay Analysis"/>
       <Attribute name="request"
                  type="http://fast.acme.com/Luxury/Request" .../>
       <Attribute name="comment" type="Text" .../>
       ...
     </Class>
  </defines>
  <extends>
     <Class url="http://fast.acme.com/Luxury/Request"/>
       <Attribute name="delay" type="String" .../>
       <Attribute name="analyses"
                  set="http://fast.acme.com/Quality/Delay%20Analysis"
        .../>
  ...
```

http://slow.acme.com/~barney

```
<Perspective name="~barney" owner="barney">
  <extends>
     <Class url="http://fast.acme.com/Luxury/Request"/>
       <Attribute name="difficulty"
                  type="enum{HIGH,LOW}" .../>
...
```

4 **http://fast.acme.com/Luxury/Task/12/***

```
<InstanceFragment id="12">
  <title>Align X with standard Z</title>
  <forecast>2011-10-10</forecast>
  ...
</InstanceFragment>
```

http://slow.acme.com/resource/Task/12/difficulty

```
<InstanceFragment id="12">
  <difficulty>HIGH</difficulty>
</InstanceFragment>
```

Figure 7-2. Example XML snippets of main read requests and associated replies

The communication protocol between components has been kept as simple and standard as possible in order to enable integration of legacy applications in a perspective-centric landscape, as we will describe in section 7.3.

7.2. End-user experience

This section presents a number of user interactions and screenshots, first manipulating *instances*, and second manipulating the *model*, illustrating a rudimentary interface for end-user modeling.

7.2.1. Instance manipulation

Instance manipulation is what corresponds to the regular daily use of present business applications. Our main objective was to verify that the dynamic, distributed nature of a perspective-centric application could be made reasonably transparent to end-users during such normal use.

The first screenshot below shows a Luxury-like official application displaying a single object.

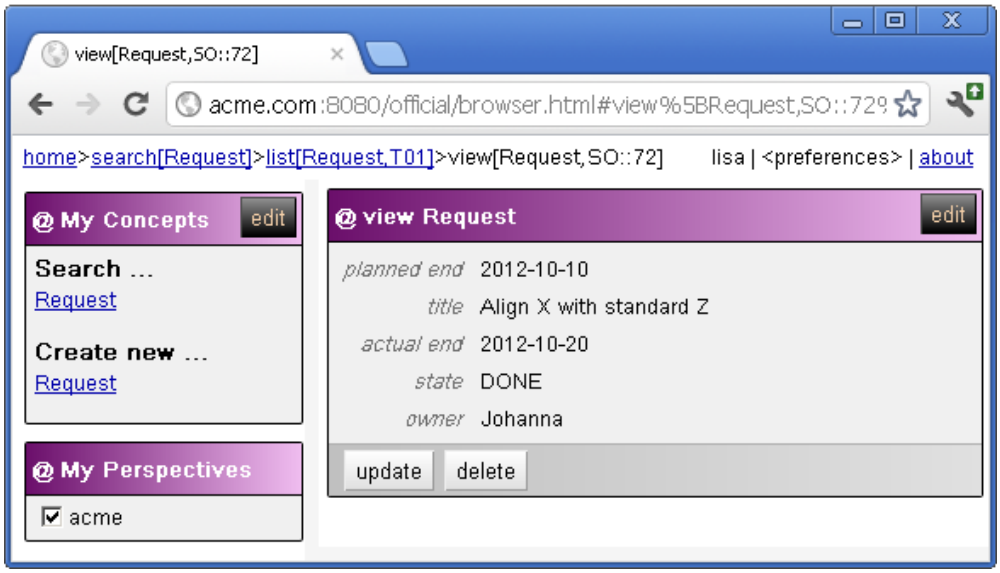


Figure 7-3. Screenshot of a perspective-centric Luxury-like official application

The screenshots below show two different users connected to the Luxury-like application, both displaying the same request object. The first user (maria) belongs to the quality group and thus sees the delay attribute and DelayAnalysis objects, whereas the second user (barney) from the planning group sees SubTask objects.

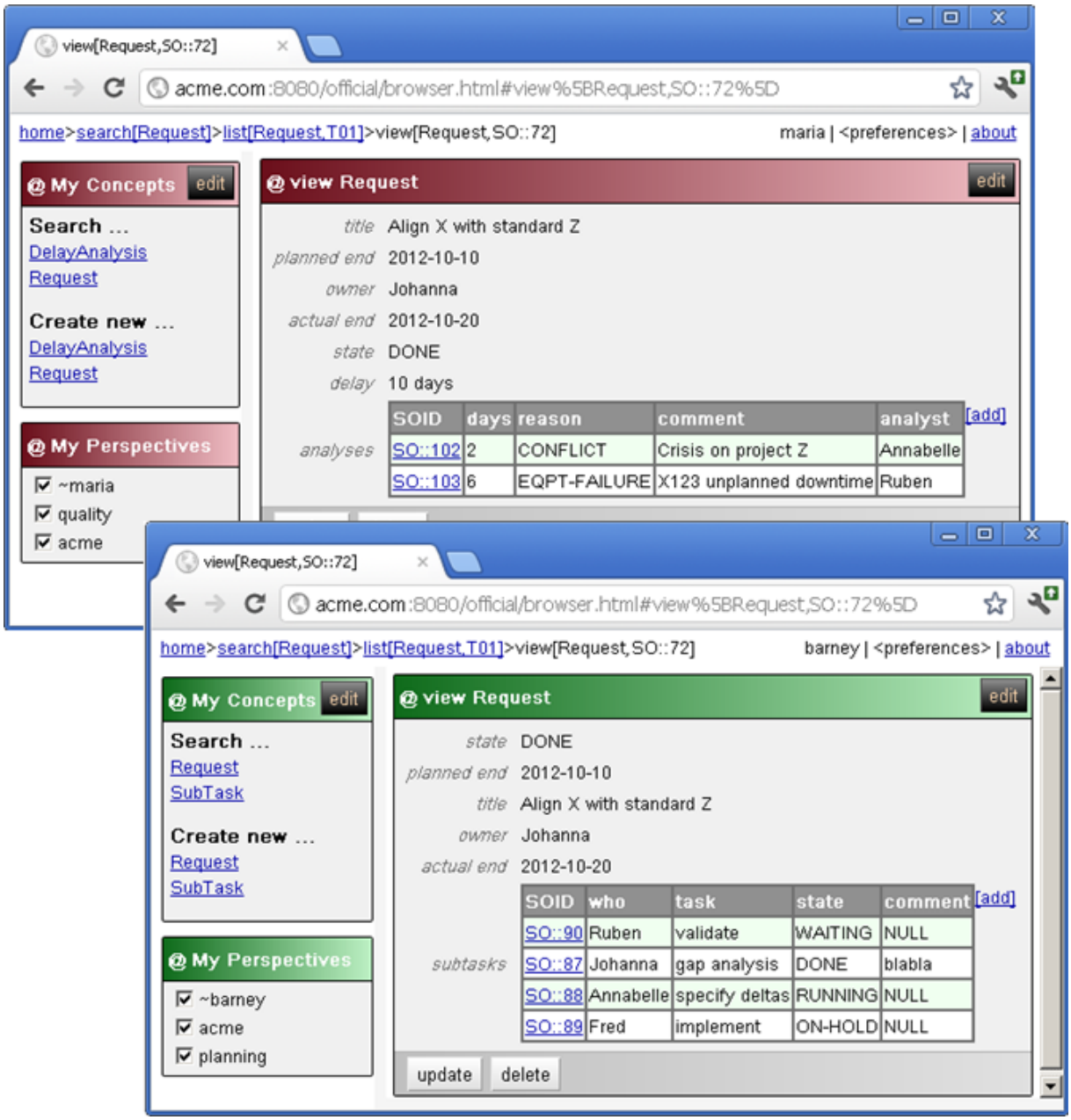


Figure 7-4. Two users with different extensions displaying the same Request object

It is important to stress again the *additive* nature of the system, as opposed to subtractive (see Figure 5-1 and Figure 5-2). In a subtractive (i.e. filtering) approach, somewhere an element would exist with all attributes, which are removed depending on the users' profile. In the prototype, multiple ClassFragments are hosted on different servers, are composed by the weaver and presented together by the browser. The same is true for InstanceFragments.

The Perspective Box

A first visible difference between the perspective-centric browser and a regular business application is the perspective box on the lower right side of the screen. It allows the user to inspect perspectives, i.e. see which fragments they define, import, extend or unimport. We think it is beneficial to raise user awareness about perspectives in this manner. An additional benefit of the perspectives box is that it allows enabling or disabling selected perspectives, as

discussed in section 6.2.1. The figure below shows a user who has disabled the “quality” perspective, which (when enabled) provides the new Element “Delay Analysis”, and extends existing Element “Request” with a new attribute “delay” and a collection of “Delay Analysis” objects.

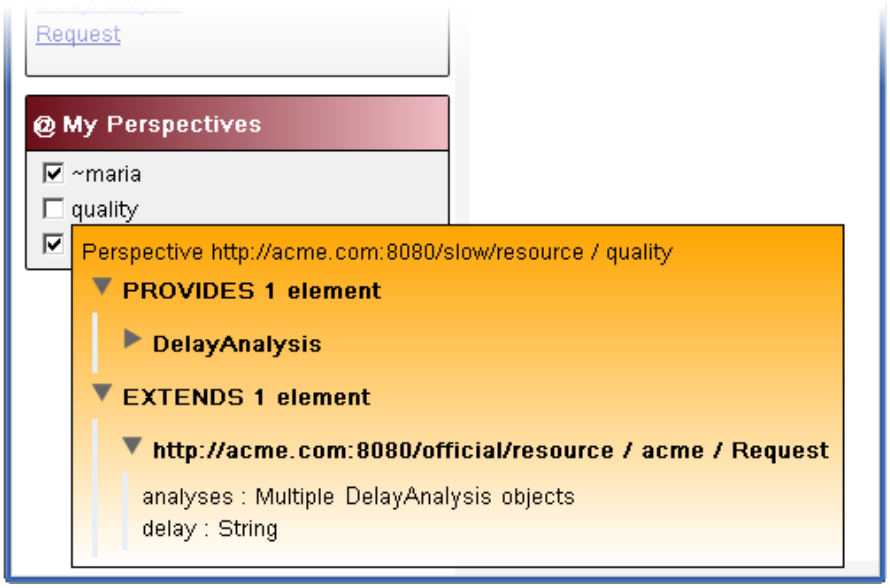


Figure 7-5. Screenshot of a user inspecting a (disabled) perspective

Asynchronicity Showing on the Surface

The other differences between the browser and regular business applications are related to the asynchronous nature of a perspective-centric system, dictated by requirement R6 (resilience). While the fragmentation of elements is transparent to the end user if the various repositories hosting the fragments have similar response time, it becomes apparent as soon as there is a perceivable difference in performance. In the remainder of this section we make the assumption that official ClassFragment "Request" is hosted on a fast and reliable server, and that the “quality” perspective has an extension of "Request", adding an attribute "delay", hosted on a slow server.

The first situation where noticeable performance differences can make the asynchronous foundation show on the surface is during the “initialization”, i.e. the time between starting the user interface and receiving the last model fragment from the slowest Repository. The screenshot below shows a user interacting with the official part of a Request object, while the extensions from the “quality” and “~maria” perspectives have not yet been received. This is illustrated by the animated arrows next to the name of the perspective[23]. The screenshot demonstrates that even when an object from an official system has been extended with fragments hosted on unreliable or slow servers, the user can still interact with the official object.

[23] The number between parentheses indicates the number of pending requests, which could be displayed in the more familiar form of a progress bar.

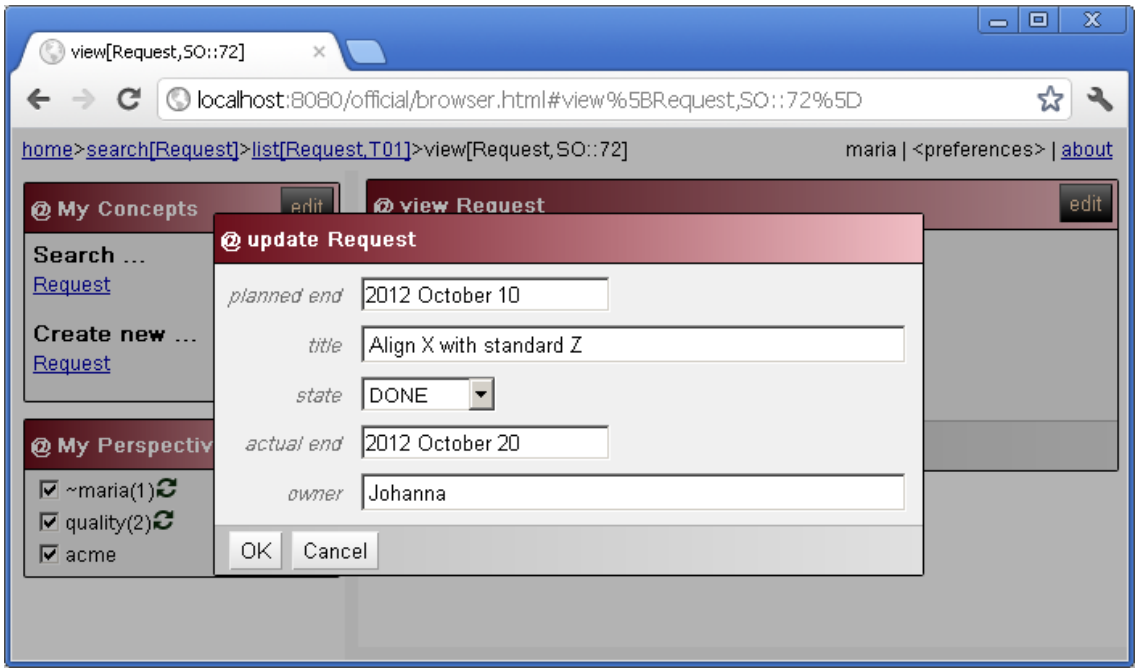


Figure 7-6. A user interacting with fragments from robust server while still waiting to receive extensions

It can be noted that the menu does not show the “Delay Analysis” extension which has not yet been received, neither does the form show the “delay” attribute for the same reason.

The menu gets updated immediately when the fragments are received, and the missing “delay” attribute will appear on subsequent forms as illustrated by the next screenshot, where the system is aware of the existence of the “delay” attribute but has not yet received its value for the current object, as indicated by the animated arrows. Once the value is received, the arrows are replaced by the regular widget for updating the value.

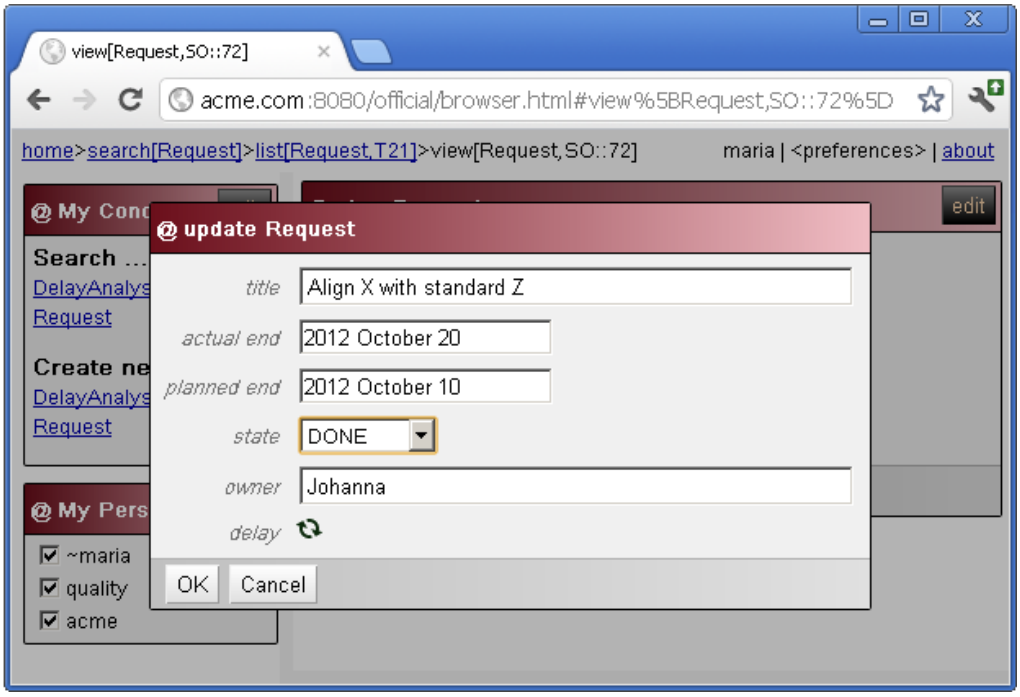


Figure 7-7. A user updating an instance in spite of missing values from a slow server

Another case when performance differences can make the fragmentation of elements visible is when a user searches for objects using criteria hosted on different servers. As an example, we can imagine a user searching for all Requests with a title containing the word “engine” and with delay equal to “1 day”. This would result in the following sequence of events.

t_0 ask repository fast.acme.com for all Requests with title="*engine*"
ask repository slow.acme.com for all Requests with delay="1 day"

t_1 get objects 8, 9, 10, 11 and 12 from fast.acme.com
*display candidate objects: **8, 9, 10, 11** and **12** (preliminary result)*

t_2 get objects 11, 12, 13, 14 and 15 from slow.acme.com
*display the intersection: **11** and **12** (final result)*

Figure 7-8. Sequence of events when searching across servers with different response times

At t_1, if we want to be meet requirement R6 (resilience), we must display the official objects regardless of the slow extension. The result of the search thus displays the list of candidate objects 8, 9, 10, 11 and 12 at t_1. Only at t_2 can the system determine that only objects 11 and 12 are valid results. It must thus be made clear to the end user that the result at t_1 is preliminary, which is not trivial. The preliminary result is a superset in the case of an "AND" request and a subset in the case of an "OR" request, and in both cases the user could take a bad decision if the incomplete nature of the result is not clear to him. The present prototype displays the preliminary result in a red font and changes to regular black font once the results are confirmed. How to clearly represent the preliminary nature of a result from a user interface point of view is beyond the scope of our study.

7.2.2. Model manipulation

A second 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 “Product”, and (2) that Element “Request” is a composition of Fragments from three different perspectives.

Figure 7-9. A user inspecting her model

Another important goal was to experiment with a first level of form-based end-user modeling. The screenshot below shows a user extending existing element “Request” with a new private attribute “customer impact” hosted in her perspective “~maria”. In terms of sequence, the operation in Figure 7-10 has occurred before the inspection in Figure 7-9.

Figure 7-10. A user extending Element “Request” with a new, private attribute

The next screenshots show the effect of this extension on the user forms, with a different widget for the search and update forms of the ClassElement “Request”.

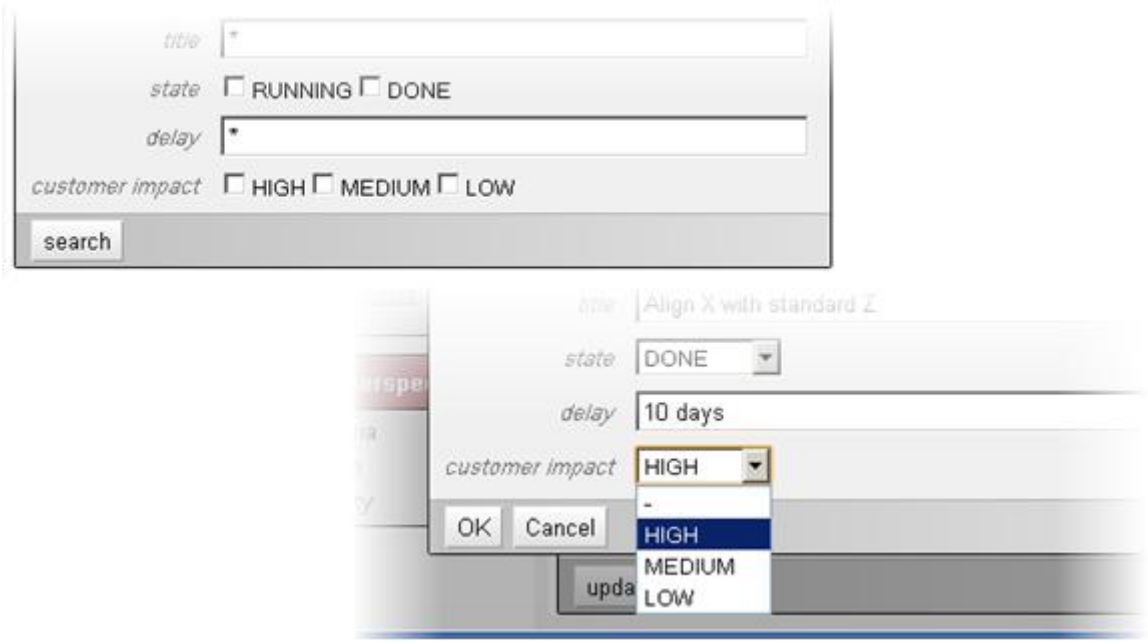


Figure 7-11. The effect of the operation in Figure 7-10 on the search and update forms

It is worth highlighting that the operation shown in Figure 7-10, i.e. adding an attribute to the “Request” Element, has *transparently* created the extension Fragment “**Request” shown in the inspection window in Figure 7-9.

As mentioned in section 3.1, 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 [149] possible.

7.3. Legacy integration

Even though our proposal is focused on a fundamentally new approach to the information system as a whole, it is important to consider how to integrate with legacy applications.

A specific implementation of the Repository component wraps a legacy system at the database level. Legacy Elements are exposed as one or several Perspectives providing top-level Fragments. This allows to selectively import and unimport elements, and most importantly to *extend* the legacy concepts. The figure below shows a user updating a "Bug" object, composed of a legacy ClassFragment, with an extension hosted on a different server adding attributes "root cause", “platform” and “notes”. The screenshot also shows that the wrapper allows read-only access to the legacy data (attributes “id” to “priority”).

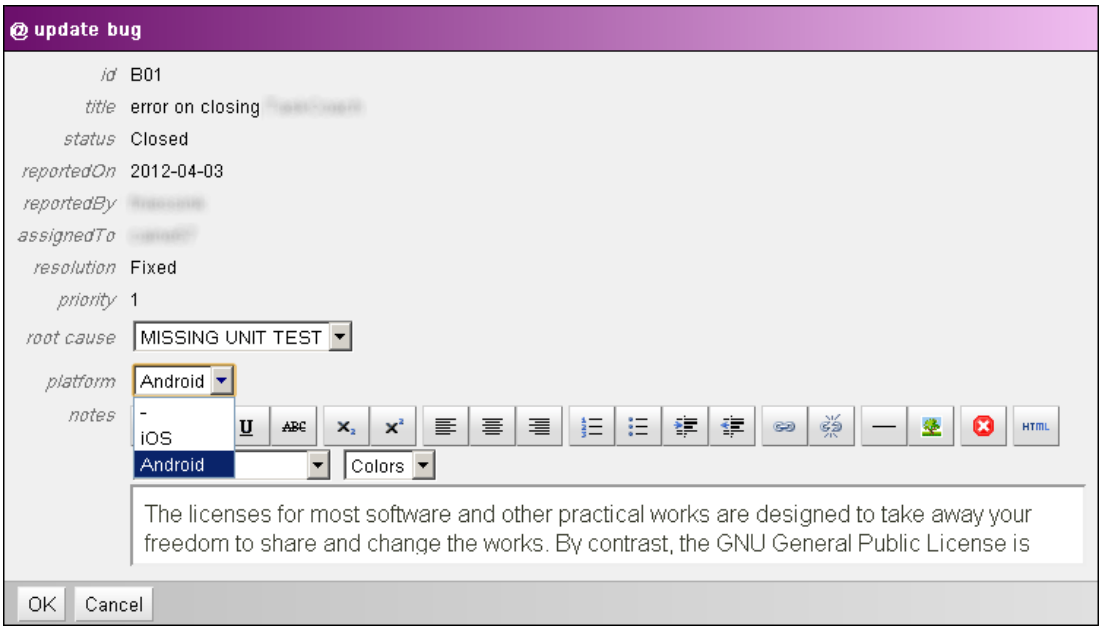


Figure 7-12. User updating a "Bug" object part legacy, part extension

The prototype wrapper allows read access at the database level. A more complete legacy integration would need access at the service level in order to not bypass the business logic layer, and write access.

In conclusion, the prototype wrapper has shown that it is possible to seamlessly integrate legacy applications in a perspective-centric information system. This wrapper has been implemented by a third party, using different technologies (Apache and PHP5) from the rest of the prototype, which illustrating the interoperability potential of the architecture.

7.4. A Prospective FeatureFragment

In order to complement our data-centric prototype, we have implemented a mechanism for invoking server-side, non-extensible FeatureFragments. In order to validate this mechanism, we have implemented the GANTT chart feature shown in the screenshot below, which has not revealed any unexpected difficulties in envisioning features as Fragments hosted by a Perspective. However, the description and generalization of FeatureFragments are beyond the scope of our study.

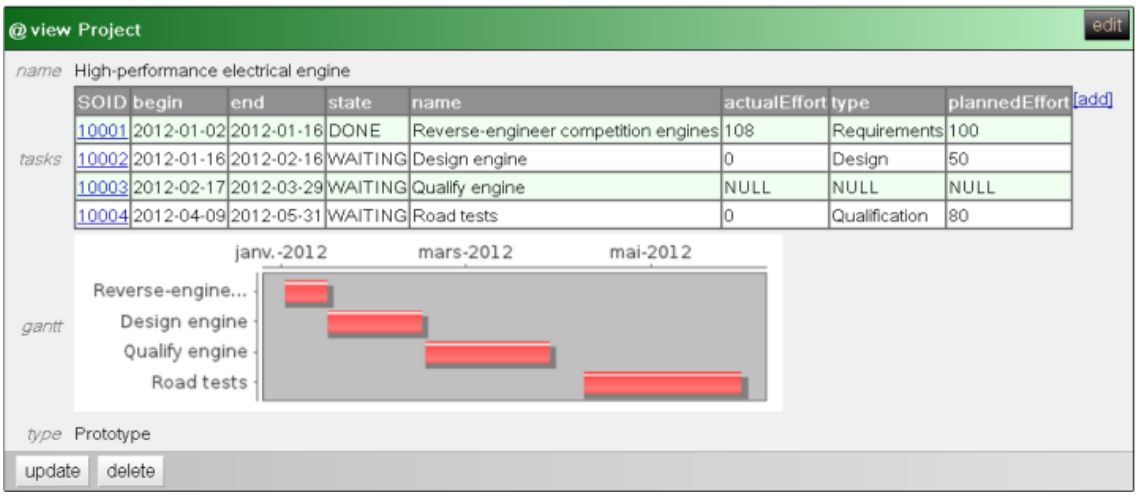


Figure 7-13. A GANTT diagram as an example of FeatureFragment

7.5. Limitations of the Current Prototype

The main limitation of the prototype is its incomplete coverage of the perspective-centric reference architecture presented in section 6. While providing the Directory, Repository, Weaver and Browser components, it is lacking the high-level components, i.e. the Registry and Monitor component. This is due to our choice to build the system from the bottom up.

The features of this available subset of components have been selected in order to enable the experiments described hereafter, excluding features which would have been interesting but not vital. Without trying to be exhaustive, we can provide the following examples of such features.

- The prototype only supports InstanceFragments hosted on the same server as their ClassFragment. It would be interesting to remove this limitation, using the Perspective as not only a vertical partitioning mechanism [146] but a horizontal one as well, undoubtedly raising new issues.
- There is no mechanism to hide, re-arrange and rename fields on a form. This feature is available in some applications today, and is fairly simple to implement in theory but it would have been interesting to measure the benefits and drawbacks of perspectives for the layout personalization as well. We can imagine FormFragments, with a specific FormElement composition logic managing the potential conflicts between multiple inherited FormFragments.

Other features are implemented in the lower layers to enable experimentation but are merely missing from the user interface, like the ability to rename and delete ClassFragments and Attributes.

Although based upon production-quality infrastructure like a JavaEE application server and a relational database system, the prototype has been designed with experimentation in mind and would require significant improvements to be deployed in a production environment.

7.6. Conclusion

In this chapter we have presented a prototype implementation of a subset of the components of the perspective-centric reference architecture presented in chapter 6.

The main goal of this prototype was to provide a concrete substrate for the experiments described in the next section.

8. Evaluation

The following sections present an evaluation of our proposal.

We describe the **experiments** based on the prototype and their results, and then provide an **evaluation** of social information systems versus our initial requirements for an agile information system. Some **performance measures** of the runtime composition mechanism are presented. We discuss **related work**, and conclude with some **challenges** and directions for future research.

8.1. Experiments

8.1.1. Simulation on Fictional Shadow Application Scenarios

As a first set of experiments, we have instantiated the prototype with various fictional industry use cases, primarily a Luxury-like application as illustrated by the screenshots from Figure 7-3 to Figure 7-13. All use cases featured multiple groups and individuals with different perspectives deployed on different servers.

These experiments with the prototype enable us to make the following assertions, which we will translate into an assessment versus the requirements in section 8.2.

- It is possible to represent business objects in the form of multiple, logically and physically distributed, actor-specific fragments
- It is possible to asynchronously compose these fragments at runtime according to the actors' profile, providing unified elements, which can be temporarily incomplete (in the presence of slow servers) but are always consistent and usable
- By applying a standard form pattern, it is possible to derive a profile-specific presentation layer from these unified elements (Figure 7-3 and Figure 7-4), providing all basic business application operations
- End-users can choose to enable and disable perspectives at runtime, with immediate adjustment of the presentation layer (Figure 7-5)
- It is possible for an end-user to dynamically change elements through forms during execution and reflect these changes immediately in all application tiers
- Legacy systems can be seamlessly integrated and adapted via a perspective-centric service facade

An important goal we had in mind was total transparency of the dynamic, distributed nature of the system for end-users. The prototype has revealed that this goal conflicts with requirement R6 (resilience) as described in the situation of distributed search with missing values (section 7.2.1). It has also revealed that it was preferable to explicitly refer to perspectives in the user

interface, in order to allow enabling and disabling them and for traceability and awareness purposes (requirements R11 and R14).

8.1.2. Simulation on Real-Life Shadow Applications

For the purpose of evaluation, the authors of [39] have provided us with additional information about Luxury, Fallen and the associated shadow applications mentioned in their paper. This has allowed us to instantiate fake versions of both official applications in our prototype.

The following perspectives have been defined to represent the relevant subset of the real-life situation, judged by the authors of [39] as "*close enough for the purposes of the evaluation*".

- One “organizational” perspective: "acme corporation"
- Two “official application” perspective: “Luxury” and “Fallen”, both providing core concepts similar to the real official applications.
- Two role perspectives: “quality” and “operations”
- One region-specific perspective: “japan”

During two remote presentations, we have been able to walk through the use cases in [39] with the first author, and have confirmed together these real-life shadow applications could have been avoided with a robust implementation of the prototype's features. The table below summarizes the results of the main use cases (shadow requirements), and mentions the main necessary improvements.

official application	shadow requirement	result	main improvements required
Luxury	subtasks	☑	consistency checking in terms of dates, efforts, and states[24]
	delay analysis	☑	same as above
Fallen	Japanese extension	☑	ability to unimport the original English attributes or at least hide them

Table 15. Results of simulation with real-life shadow applications from [39]

This experiment shows that a robust implementation of a perspective-centric architecture technically removes most needs for introducing shadow applications. This is not proof that this would actually have happened in real life, but represents a conclusive result nonetheless.

The experiments above cover the use cases reported in [39], i.e. shadow applications providing additional concept and attributes, and shadow applications providing an additional level of decomposition, which the authors consider as the majority of shadow application requirements. An important third scenario they have mentioned after the presentations is *“pulling together data from various systems, official or not, and presenting this data in a*

[24] The bottom screenshot of Figure 7-4 (page 85) shows an example the need for consistency checking rules. The Request has a state “DONE”, while some SubTasks are still “RUNNING” or “ON-HOLD”.

unified way with potential additional data", which is possible in the prototype implementation but in the absence of more detailed Boeing use cases has not been included in the presentation.

8.1.3. Acceptance of Perspective-Centric Concepts

In order to evaluate the assumption that end users can adapt a business application to their needs through forms, we have conducted an experiment where a focus group [150] of 3 subjects was requested to play 3 different roles inspired by our reference use cases from [39].

A fake official application was provided, with 4 concepts. Subjects were handed a role sheet which explained their job, for which the official application was only partially suited, and they were asked to adapt the official application to their needs for 30 minutes. For all 3 roles, successfully adapting the application involved to import an already available fragment, extend available fragments and introduce a new element themselves. A complete description of the experiment is provided in appendix F. The figure below illustrates a role sheet, the initial official application (i.e. the state at the beginning of the experiment) and the adapted application (at the end of the experiment).

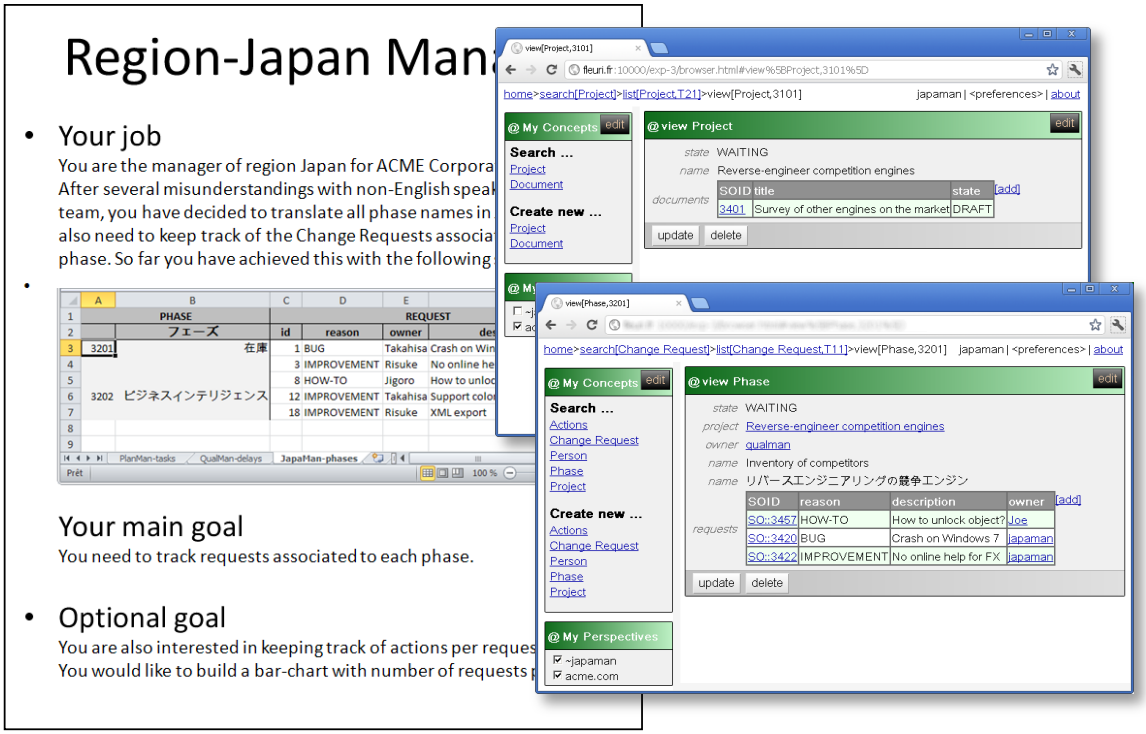


Figure 8-1. Example role sheet, and application before and after the experiment

We have run this experiment 4 times, i.e. with 12 subjects; all are at least occasional users of business applications. 6 were industry employees, 6 academic staff; in each category, 3 were professional software developers, 3 were not, although all subjects were familiar to some extent with business application management or configuration. We have excluded "pure" end-users from this experiment due to the present modeling interface which requires a degree of software literacy, and which we do not consider intuitive enough for users unfamiliar with concepts like Attribute and Type (see screenshot in Figure 7-10).

In spite of the rudimentary modeling interface, *all 12 subjects have successfully completed the experiment*. The detailed results of the 4 runs are provided in appendix F.

In the groups of subjects who were not software developers, some subjects have asked for guidance on how to add *associations* between the central concept ("Phase") and related concepts like "SubTask" and "DelayAnalysis", which we think is due to the difficulty of mentally mapping associations with the example spreadsheets provided in the role sheets.

In order to conduct experiments with broader and more diverse audiences, more work on the intuitiveness of the modeling interface is required, with particular attention to the representation of associations between concepts.

8.1.4. Qualitative Prototype Feedback

In addition to the aforementioned experiments, we have formally presented an early version of our prototype to 8 information system professionals, individually, 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. 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.

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 persistence tier. 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 later*", i.e. once the application has been collaboratively built and validated. This directly contradicted his earlier statements about continuously evolving and conflicting requirements, which he has acknowledged.

The final version of the prototype has also been demonstrated to the corporate IT architecture group (6 senior architects) of a major European high-tech company of over 100 000 employees. As a conclusion, they have stated that *"if a new official application (or a new version of an existing official application) provided (a mature implementation of) these social extension capabilities, this would avoid (or strongly diminish) the emergence of shadow applications"*[25]. Their concerns were not related to the concepts or the technology, but to communication and training, i.e. changing decades of habits of people "helping themselves".

Besides these two initiatives to collect qualitative feedback, we have informally presented our proposal and prototype to dozens of information system professionals, always receiving encouraging feedback. We would like to mention the following two concerns that were often raised early in the discussion.

[25] "Si une nouvelle application officielle (ou nouvelle version d'une application officielle existante) fournissait (une implémentation mature de) ces capacités d'extension sociale, elle pourrait éviter (ou limiter fortement) l'apparition d'applications périphériques"

- When all employees from a multi-national corporation can contribute fragments to the information system, *duplications and inconsistencies* are guaranteed.
- Information system *governance* appears impossible in an environment where all components change continuously.

First, we think both concerns share an incomplete assessment of the present situation, where duplications and inconsistencies are already a problem and governance is restricted to a small subset of applications at best. Today these problems are impossible to measure and thus easy to underestimate or even ignore. Second, we consider that the awareness and monitoring mechanisms we propose provide assistance in preventing and resolving inconsistencies respectively, and thus think governance would become easier, not more difficult.

Another interesting objection has emerged three times with different individuals and companies: *"great idea, but this will never work here (in our company)"*. In all three cases, the persons' explanation was that political tensions and internal competition would keep actors from sharing anything with their colleagues. It is tempting to dismiss these objections as coming from dysfunctional corporations with poor management and an unhealthy corporate culture (which no architecture paradigm can cure), but the problem may be common and serious enough to warrant further study.

8.2. Evaluation versus Requirements

In this section we evaluate perspective-centric architectures versus the requirements for agile information systems presented in chapter 3. We will rank different levels of our proposal.

- We consider that the proposal meets the requirement at a *conceptual* level if the high-level conceptualization presented in chapter 5 provides a theoretical solution.
- If the reference architecture presented in section 6 describes a solution to a requirement, we consider it is met at the *design* level.
- Our strongest claim for meeting a requirement is when the prototype *implementation* and associated experiments presented in chapter 7 have demonstrated a solution.

When an actor A faces a new business requirement which needs to be reflected in a given application, the prototype has shown that regardless of who owns the application, actor A can adapt it himself. His adaptation is hosted in a separate perspective which isolates his change from both the initial application and other adaptations. Thanks to perspectives, actors thus have freedom to adapt applications when needed, without disrupting other actors, which satisfies both requirements **R0** (influence) and **R1** (isolation).

The distributed nature of the prototype has shown that it is possible to host adaptations on hardware resources different from the original application if any. Business-units can thus introduce the hardware resources they need for their specific requirements, satisfying **R2** (hardware independence).

The prototype has demonstrated the feasibility of simple form-based end-user adaptations of applications. More complex adaptations are possible by providing a spreadsheet-like scripting language. We can thus consider that **R3** (ease-of-modification) is met.

Distributed repositories isolate fragments, allowing their owner to decide who they grant access to. The prototype implementation does provide this isolation but does not have an access control mechanism, and we thus consider requirement **R4** (confidentiality) as met at the design level.

The table below summarizes the rankings of perspective-centric concepts, reference design and prototype implementation versus application-level requirements R0-R4.

		perspective-centric architecture		
requirement		**conceptualized**	**designed**	**implemented**
R0	influence	☑	☑	☑
R1	isolation	☑	☑	☑
R2	hardware independence	☑	☑	☑
R3	ease-of-modification	☑	☑	☑
R4	confidentiality	☑	☑	

Table 16. Ranking of perspective-centric architectures versus application-level requirements

Like service-oriented architectures, our prototype implementation has the capability to adapt official fragments with actor-specific aspects without duplicating the initial fragment. We can thus consider that the global information system consistency is preserved and that requirement **R5** (consistency) is met.

Unlike service-oriented architectures, adaptations do not encapsulate the initial element but merely complement them. As a result, a slow or missing adaptation does not break the communication between the client and the official element. The prototype has demonstrated the ability to interact with available elements even when adaptations are unreliable (Figure 7-7), and we thus consider requirement **R6** (resilience) as met.

The prototype's fragment composition mechanism traverses the actor-perspective graph and composes a unified model including all fragments relevant for the current profile, meeting requirement **R8** (profile-driven composition). This allows the Browser to present a single profile-specific user interface to the end-user, providing all and only relevant elements. Requirement **R7** (uniformity) is thus satisfied as well.

		perspective-centric architecture		
requirement		**conceptualized**	**designed**	**implemented**
R5	consistency	☑	☑	☑
R6	resilience	☑	☑	☑
R7	uniformity	☑	☑	☑

Table 17. Ranking of perspective-centric architectures versus information system-level requirements

During our experiments (section 8.1.1), we have been able to verify that changes in a high-level perspective, whether these changes are evolutions of existing fragments, introduction of new fragments or even deletion of fragments are reflected in all relevant users' runtime environments. In the present implementation, this happens at session initialization-time only, but in principle the asynchronous model composition mechanism can reflect the changes immediately, adding and removing elements on the fly. We thus consider requirement **R9** (change propagation) as satisfied.

The Registry component allows the owner of adaptation A' to publish it, making the adaptation visible to the owner of initial fragment A. The owner of A is thus aware of all published adaptations of his element, and can analyze the impact of a change on the overall system before performing the change. This meets requirements **R10** (forward traceability).

Figure 7-9 (page 98) shows a user inspecting her model. She can clearly see that the element "Request" she is manipulating is composed of several fragments, and determine their respective origins. This helps in getting support (explanations) at the right level, and improves the end-user understanding of the elements on her screen. We can thus consider that the prototype provides a mechanism for backward traceability which satisfies requirement **R11**.

		perspective-centric architecture		
requirement		**conceptualized**	**designed**	**implemented**
R8	profile-driven composition	☑	☑	☑
R9	change propagation	☑	☑	☑
R10	forward traceability	☑	☑	
R11	backward traceability	☑	☑	☑

Table 18. Ranking of perspective-centric architectures versus traceability and composition requirements

The Registry component provides a way for all actors to share their elements with other actors, whether a single individual, groups, or even the entire corporation. Like service-oriented architectures, the Registry component provides a central referential for all available fragments, which meets both requirements **R12** (sharing) and **R13** (awareness). The social mechanisms described in section 5.2 provide a promising solution for sorting available fragments according to their relevance for a given user, meeting requirement **R14** (relevance).

		perspective-centric architecture		
requirement		**conceptualized**	**designed**	**implemented**
R12	sharing	☑	☑	
R13	awareness	☑	☑	
R14	relevance	☑		

Table 19. Ranking of perspective-centric architectures versus traceability and composition requirements

In a successful perspective-centric information system, all fragments and their dependencies are well known. We have described the expected benefits of perspectives with respect to governance, in terms of observation (monitoring), community management and inconsistency

detection and resolution, and will thus consider that a perspective-centric information system meets requirement **R15** (governability).

	perspective-centric architecture		
requirement	**conceptualized**	**designed**	**implemented**
R15 governability	☑		

Table 20. Ranking of perspective-centric architectures versus governance requirements

The table below summarizes the rankings of application-centric architectures, service-oriented architectures and our perspective-centric proposal. It illustrates how a perspective-centric system successfully borrows the benefits from both shadow applications and services, and leverages the aforementioned benefits of model-driven engineering, end-user development and social computing to provide a promising paradigm fulfilling all our requirements for an agile information system.

Category	**Requirements**	**Application-centric architecture**	**Service-centric architecture**	**Perspective-centric architecture**
Application	**R0**: Influence	☑	☑	☑
	R1: Isolation	☑	☑	☑
	R2: Hardware Independence	☑	☑	☑
	R3: Ease of Modification	☑	☑	☑
	R4: Confidentiality	☑	☑	☑
Information System	**R5**: Consistency	✗	☑	☑
	R6: Resilience	☑	✗	☑
	R7: Uniformity	☑	☑	☑
Composition	**R8**: Profile-driven Composition	✗	✗	☑
	R9: Change Propagation	✗	✗	☑
Traceability	**R10**: Forward Traceability	✗	✗	☑
	R11: Backward Traceability	✗	✗	☑
Collaboration	**R12**: Sharing	✗	☑	☑
	R13: Awareness	✗	☑	☑
	R14: Relevance	✗	✗	☑
Governance	**R15:** Governability	✗	✗	☑

Table 21. Rankings of the various architecture paradigms versus requirements for agile information systems

8.3. Performance Measures

It appears obvious that dynamically composing a model at runtime instead of building a static model at development-time has a performance impact. However, the manipulation of the prototype on the example uses cases has no perceptible influence, which we attribute to the continuously growing processing power and network bandwidths versus the fairly simple and stable core requirements of business applications.

Beyond this subjective observation, this section discusses two performance aspects of our proposal. First, we measure the performance of the runtime model composition mechanism, and second, we discuss the performance perception from the end-users' point of view.

8.3.1. Performance of the Runtime Model Composition Mechanism

In order to evaluate the performance impact of runtime model composition, we have measured the time to compose one single concept with 100 attributes. The table below shows a few composition scenarios for these 100 attributes.

number of perspectives	number of root fragments	number of extension fragments	number of attributes per fragment
1	1	0	100
10	1	9	10
100	1	99	1

Table 22. Example composition scenarios for one concept with 100 attributes

It is important to remember that composition is driven by the current users' profile, and that the scenario of 1 concept having 99 *relevant* extensions for the current user is extremely unlikely. We estimate that the highlighted case of 10 perspectives (i.e. 1 root fragment and 9 extensions) is representative of a maximum number of relevant extensions in real-life situations.

The execution times below have been measured on a single machine hosting the directory and repository components (with associated databases and application server) and the browser component. They include (local) network requests, database access, serialization and deserialization, model composition and browser rendering. The chart below plots the total composition time versus the number of perspectives.

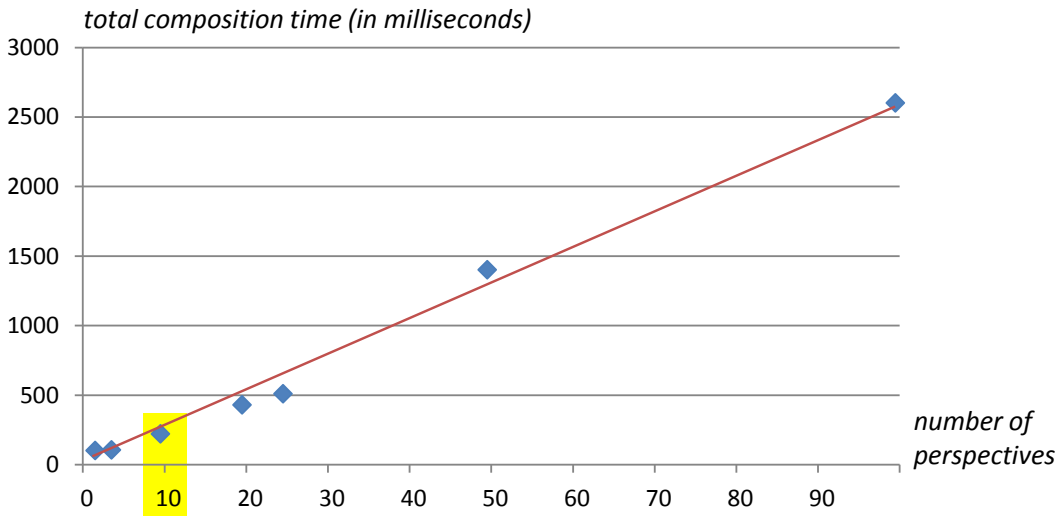


Figure 8-2. Total model composition time for 1 concept defined across 1 to 100 perspectives

The chart illustrates that in an implementation with no optimization, the performance is a linear function of the number of perspectives. Optimization can be envisioned at many levels, especially caching techniques should prove effective considering that model changes are less frequent than instance changes. Also, the asynchronous composition fetches perspectives in parallel, which in the presence of multiple physical machines should result in better performance than shown above, where hosting all servers on the same machine cancels out the benefit of parallelism.

We have not measured the performance of *instance* composition, which is the same situation as any other distributed system.

8.3.2. Performance from the End-Users' Point of View

Although not strictly a matter of performance, it is important that from an end-user point of view, a perspective-centric information system provides him with the optimal composition of fragments for his profile (i.e. his profile-specific application).

In complex corporations, the high number of possible profiles mentioned in section 2.2 makes this impossible to achieve when composition is not automatic. End-users must thus juggle with multiple applications to perform a simple business transaction in order to keep the various applications aligned.

Although we need more experiments to back up this claim, we think profile-specific applications have the potential to significantly speed up the typical knowledge workers' daily work.

8.4. Limitations

The conceptualization of a complex system involves numerous conscious and unconscious decisions. In this section, we present the most important conscious limitations part of our proposal.

We consider the main conceptual limitation of our study is its focus on *data-centric applications*. Within data-centric aspects and whether at the conceptualization, design or implementation phase, we have restricted our study to *class fragments* and the associated class elements. Both the introduction of higher level concepts like package fragments and finer-grained fragments would unquestionably raise versioning issues [151] which are not addressed by the present study.

We have described a conceptual solution to the management of big numbers of user-contributed Fragments, but have no experimental data to validate the *applicability* of consumer-space social mechanisms in the domain of end-user enterprise development.

A majority of todays' distributed business applications rely on *transactions* guaranteeing the ACID constraints [152] through synchronicity [153]. Following the CAP theorem [154], such systems guarantee consistency but not availability. In order to meet R6 (resilience), our proposal relies on asynchronous communication, which implies that a different consistency model is needed, like *session consistency* [154], a variant of the *eventually consistent* strategy. Our study has not covered the impact of such a consistency model on the design. The highly asynchronous nature of our proposal may raise other technical issues which need further work.

Other important aspects of enterprise computing have been excluded from our study due to our initial assumption that present solutions would be applicable in a perspective-centric system. *Security* is an example of such a concern. At the Repository-level, authentication and authorization requirements are indeed similar to present service-oriented architectures. However, at the Weaver-level we can imagine malicious extensions being composed with sensitive enterprise data, revealing the need for a robust security model.

8.5. Comparison to Related Work

In this section we compare our proposal for a perspective-centric information system to a selection of research domains and practices.

Social Software Engineering

The new though broad field of *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 [93]. Our proposal clearly belongs in this field [14].

Social Computing and Social Informatics

Social Computing and *Social Informatics* both designate the emerging research field concerned with the interactions between technology and society, more precisely between computational systems and social behavior. It is a highly multi-disciplinary field, which leverages previous

work in sociology, social psychology, organization and communication theory, human-computer interaction and computing. Social informatics studies claim that information technology and society *influence each other* [155]. [156] has identified several definitions, and proposes the following summary.

> *"Computational facilitation of social studies and human social dynamics as well as the design and use of ICT technologies that consider social context"*

Our proposal clearly falls in the second category. The research within the field focuses on machine learning techniques [157], and the following application areas.

- computer supported online communities, also referred to as social network sites (SNSs)
- intelligent agents in interactive entertainment
- business and public sector applications and forecasting systems

Within the business applications category, the contributions focus on recommender systems (for the purpose of suggesting products to customers) and the associated feedback mechanisms, and on forecasting (predicting sales) [156], and not on the social aspects of the internal enterprise information system. However, most research topics in social computing (motivation for participating, reputation, network effects, governance structures, intellectual property rights... [158]) directly apply to our proposal.

Linked Data and the Semantic Web

The Semantic Web is an extension of the current web in which information is given well-defined meaning [159]. Within this vision, the *Linked Data* initiative [160] aims at enabling the evolution of the present World-Wide Web of linked documents to include linked data entities. It proposes a set of practices for publishing and connecting structured data, through the following rules [161].

- Use URIs as names for things
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using standards
- Include links to other URIs so that they can discover more things.

At a high level, we can consider that both LinkedData and our proposal aim at integrating distributed, loosely coupled and independently managed repositories of persistent entities, which users then manipulate through a dynamic user interface built using mostly standard web technologies. However, there are a number of important differences between the LinkedData initiative and a perspective-centric enterprise architecture, which are summarized in the table below.

	Linked Data	Perspective-Centric System
focus	internet, mostly-read, (existing) semantic data	enterprise, read-write, business applications
goal	integration, "use the web as a single global database"	evolution, business agility, prevent shadow app chaos
central component	search-engine	directory
data	public, loosely structured	limited visibility, strongly structured
schema	global, managed by experts	actor-specific, layered, contributed by end-users
main "same entity" relationship	same-as	extends
duplicates resolution	semantic, i.e. tolerate duplicates, annotate with same-as	physical, i.e. factorize in common fragment with different extensions
functionality	out of scope	mandatory
scalability requirements and solution	extreme (world-wide), no solution	medium (enterprise-wide), perspectives provide factorization

Table 23. Overview of the differences between Linked Data and a perspective-centric information system

Further study is required to more accurately assess the overlap and complementarity of social information systems and Linked Data.

Organic Information Systems

Organic computing is an emerging field of research which aims to apply principles of biology to software systems. [3] mentions the following definition, where systems are *"… organic if all of its components and subsystems are well coordinated in a purposeful manner. Organic structures realize themselves as hierarchically nested processes, structured such as to be able to meet upcoming challenges by goal-oriented reactions"*, and highlights how this definition relates to the concept of business agility.

Component-Based Software Engineering

The Component-Based Software Engineering (CBSE) [41, 145] community is actively researching robust dynamic systems, where components can appear and disappear during execution. It provides foundation concepts and technologies for making a perspective-centric information system cope with dynamic fragments and services of variable reliability.

Other Related Work

The nature of our proposal is such that many other fields of research and practice could be considered related.

We have mentioned *ViewPoints* in section 4.4, which allow capturing and representing divergent concerns at the specification and design level. We think our proposal is complementary, and that perspectives could be considered a runtime representation of viewpoints.

Likewise, the *Requirements Engineering* community has recently started to study how to apply social mechanisms like voting and commenting to requirements gathering and prioritizing [46]. Our proposal can be considered an extension of this principle, applying the same voting mechanisms to the implementation of the requirements as well.

We consider our social information systems a new combination of numerous existing approaches, and will thus not try to further enumerate all potentially related fields.

8.6. Challenges and Further Work

In this section we present some of the challenges and directions for future work, in terms of conceptualization, usability, evaluation and corporate culture.

8.6.1. Conceptualization Challenges

Overcoming the limitations described in section 8.4, like the management of composite fragments, the integration of an appropriate transaction model and robust security, can leverage solid prior work from a number of research topics but may raise other conceptual challenges in when envisioning a high level of fragmentation and asynchronicity.

Inheritance is an important construct in data modeling. It needs to be introduced in the meta-model, and the differences and similarities between the concepts of inheritance, extension and views must be formally described.

Many other potentially useful extensions of the meta-model can be imagined, like *derived associations*. Finding and keeping the right balance between a naïve and an over-engineered meta-model is a challenge in itself.

Beyond data-centric fragments, the introduction of *functions* in our meta-model is not easy. Function overloading has been intensely studied in the object-oriented software development domain, and the more recent software composition and aspect-oriented software development domains contribute interesting prior art as well. Functions are typically expressed in *imperative* languages, which are not easily reconciled with an asynchronous model composition mechanism. A fine tracking of dependencies is necessary to determine whether all fragments required for a given function are available, and to allow its execution.

8.6.2. Usability Challenges

End-user modeling and scripting is still a topic of research in itself. In our case, significant work is necessary to blend the unfamiliar model evolution features into regular business application interfaces and provide a degree of modeling by example.

Both traceability and social features must also find the right balance between intuitive access and non-intrusiveness.

8.6.3. Evaluation Challenges

One of the main challenges of our work is to find suitable ways to evaluate the concepts underlying social information systems in a real enterprise setting. A standard approach would be to deploy a perspective-centric application with a small group of users, and study its usage. However, the prototype perspective-centric system is designed to alleviate the need for shadow applications. 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 discomfort of the IT professionals about this architecture, this requires a significant shift in thinking by both IT and business-unit managers about how crucial business data is stored and managed.

8.6.4. Cultural Challenges

Beyond successful evaluation, deploying a perspective-centric application for production usage requires to convince the corporate IT department to relinquish their present control monopoly. First, they must admit that they cannot manage the evolution of applications themselves in timely fashion. Second, they must trust the employee base to contribute relevant elements and manage the evolution of the information system as a community. Both aspects represent a challenge in the conservative world of business computing.

Besides reluctance from corporate IT departments, business units themselves may be afraid of officially taking ownership and responsibility, which is an entirely different matter than discreetly developing shadow applications.

Introducing a perspective-centric system thus requires not only technological innovation, but also a high degree of organizational open-mindedness.

9. Conclusion

We consider this study an original combination of existing approaches, proposing to apply principles proven effective in the consumer space to the conservative domain of enterprise information systems. This chapter summarizes our contributions and discusses short and long term perspectives.

9.1. Summary of Contributions

The aim of this thesis was to study how information systems could be made more agile.

Building upon the observation of the present state of information systems in big corporations, we have provided a characterization and analysis of shadow applications (chapter 2), and have proposed a generalization of their interesting characteristics in the form of a set of high-level requirements for business agility (chapter 3).

We have evaluated the two dominant information system architecture paradigms with respect to these requirements (chapter 4), highlighting strengths and weaknesses and demonstrating the need for an alternative enterprise architecture principle.

We have proposed such a principle, based upon a fundamentally different distribution of information system responsibilities, and have presented the foundation concepts of isolation through perspectives and fragment composition. We have discussed our broader vision of a social information system leveraging the collective intelligence of an organizations' employees, and the possibility of social evolution (chapter 5).

We have described a possible perspective-centric architecture, based upon a set of components for which we have provided a high level design and brief discussion (chapter 6). A subset of these components has been implemented in a prototype which has been presented (chapter 7).

The prototype implementation has allowed a number of experiments on both fictional and real-life use-cases, which have been presented along with an evaluation of the concepts, architecture and implementation versus our initial requirements for business agility (chapter 8).

A paper describing an early stage of the proposal [14] has been presented at the 3rd international workshop on Social Software Engineering in 2010, and a more complete paper summarizing our work [162] has been presented at the 14th international conference on Enterprise Information Systems (ICEIS) in 2012.

9.2. Perspectives

An immediate application of our proposal would be to embed actor-specific extension capabilities in present business applications. By allowing actors to implement their specific requirements within the application, this should already avoid the emergence of simple shadow applications in their vicinity. If the communication protocol used by such an application is well documented, legacy applications could be easily integrated by wrapping them as external perspectives.

A more advanced application would be the specification of a standard interface and communication protocol for Repository components. This would allow to design application modules independently from each other, and to leave their integration to an external perspective. In similar fashion to web and LinkedData browsers, a standard protocol would enable the development of universal client components.

The specification of standards for perspectives and fragments could in turn lead to the fragmentation of the business application market, and to increased competition, an important driver for innovation. This may not be in the interest of established enterprise market leaders who thrive on critical mass and captive customers. But it would make a place for smaller, more innovative players, which should benefit the industry as a whole.

The social feedback mechanisms we have discussed with an intra-corporation scope could be envisioned at a global level as well, beyond corporate boundaries in similar fashion to present consumer app-stores. “Fragment stores” could emerge out of this, with contributions from historical market leaders, challengers, niche players, integrators and open source communities. Perspectives could help pave the road for the old dream of a component market.

Even corporations which are not in the enterprise software market could decide to share their internally developed fragments, to propose them for adoption by a community, leveraging their collective intelligence and energy and share maintenance costs.

Our proposal could both benefit from and contribute to progress on cloud computing infrastructures. On one hand, cloud computing provides an ideal platform for collaboration across complex supply chains, sharing selected fragments with relevant partners. On the other hand, perspectives could prove an interesting solution for multi-layered multi-tenancy. Additionally, our distributed architecture and runtime composition allows to blend public data, data shared with partners in the cloud, and confidential on-premise data, potentially overcoming a reluctance[26] to cloud computing adoption.

The recent desire of individuals to bring the devices they use at home into the workplace [163], called the Bring-Your-Own-Device vision [164], implies a greater variety of terminals with, on average, smaller screens. As a result, employees need high flexibility in selecting the most important information, for which perspectives could provide an interesting support.

[26] In the case of legal obligation to keep some data on premise, or situations where corporations consider their security level superior to their cloud platform providers’.

From a software engineering point of view, perspectives could help in integrating running development projects with live production environments, facilitating continuous integration and delivery. Boundaries between mockup, prototype, beta and production environments could be smoothened and concurrent development made easier, as well as experimentation encouraged.

9.3. Conclusion

In [20], Barry Boehm provides an overview of the history of software engineering by adopting the Hegelian view that increased human understanding follows a path of *thesis* (a substantiated proposal); *antithesis* (the thesis fails in some important ways; here is a better explanation); and *synthesis* (the antithesis rejected too much of the original thesis; here is a hybrid that captures the best of both while avoiding their defects).

Within the domain of information systems, we have considered present enterprise applications as the thesis (1960-present) and shadow applications as the antithesis (1990-present). Our social information systems proposal thus aims at a contribution to a synthesis, preserving the strengths of both and resolving their weaknesses by leveraging recent progress on model-driven engineering, end-user software development, service-oriented architectures and social software engineering.

Appendixes

A. Disambiguating the term "Application"

Our study targets business software applications for human users, excluding machine-to-machine systems (reservation backends, ...). We also exclude system software like database management systems (DBMSs), application servers and other kinds of middleware.

In this section we try to disambiguate the term Application (both from this human user's point of view and from a technical / architecture point of view), and introduce the main roles. The following diagram shows a high-level overview of the application lifecycle, defining the main phases, roles and artifacts, each described in more details in the next sections.

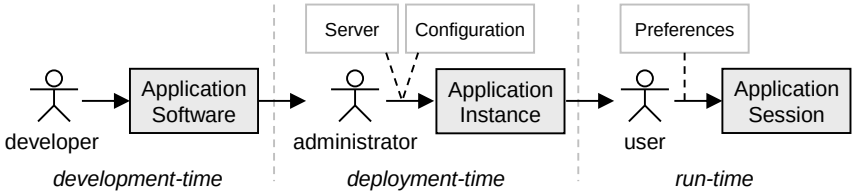


Figure 0-1. High-level overview of the Application lifecycle

The figure above introduces the following artifacts, all of which are commonly referred to by the term Application.

- *Application Software* is a set of executable programs and other resources (configuration files, documentation), typically identified by a name and version
- *Application Instance* = application software installed in a runtime environment and configured to solve a specific business problem for a given organization
- *Application Session* = what happens in front of a users' eyes

Even at such a high level, the social aspects of business software are apparent. Although in real enterprise settings a huge number of roles exist, we can consider the following grouping by their impact on the various artifacts.

- *Developers* are actors (with influence on)/(whose decisions are embedded in) the application software. Examples are product managers, architects, programmers, testers, ...
- *Administrators* are actors deciding how to configure an application instance : integrator, consultant / application engineer, functional administrator, system administrator, ...
- *Users* are actors effectively using the application to perform their job

Development-time: Development of Application Software

At development-time, Application Software can be considered a collection of coarse-grained *Elements*. Below are a few examples of Elements.

In a Project Management application, the business concepts "Project" and "Task" can be considered Elements. They represent persistent entities of the problem domain, and can involve a number of technical representations: a database table "Task" with the associated database triggers, various object representations like a JavaEE EntityBean "Task", etc.

In the same application, features like the critical path calculation algorithm or the GANTT chart can be considered elements as well, which rely on the aforementioned business concepts.

There are many ways of envisioning Application Software development, and thus of deciding which Elements are implemented. As an example, a developer can provide a single “generic” Element which produces the forms required to edit a Project, a Task, etc. Or he can decide to implement specialized forms EditProject and EditTask.

Most Elements are static. Some Elements are configurable, i.e. designed to be further refined at deployment- and/or run-time. The level of configurability varies from very low (typical for in-house ad-hoc software developments) to fairly high for generic applications like Product Lifecycle Management systems.

Application Software packages usually define Permissions restricting access to Elements, thus defining logical subsets.

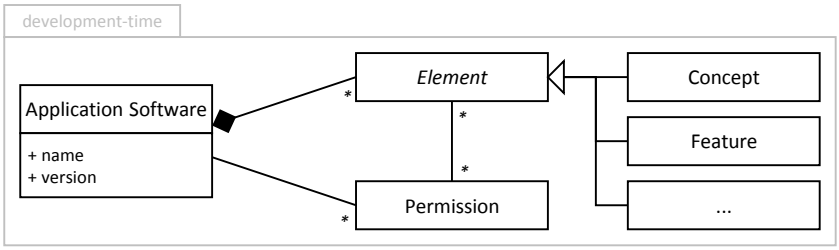


Figure 0-2. Application Software at development-time

In terms of artifacts, Application Software is a set of executables and related resources like scripts, configuration files and documentation. They are typically identified by a name and version, for example Microsoft Enterprise Project Management version 2010.

The resulting application reflects the decisions of a (typically fairly small [46]) group of software developers.

Deployment-time: Configuration of Application Instances

At deployment-time, the goal is to solve a specific problem for a given organization. An Application Instance is created by physically installing Application Software in a runtime environment and by using Element configurability to adapt it to the specific purpose. Actors (i.e. Users or Groups) are granted Permissions. Application Instances reflect the decisions of yet another small group of software integrators and administrators - constrained by the earlier decisions of developers.

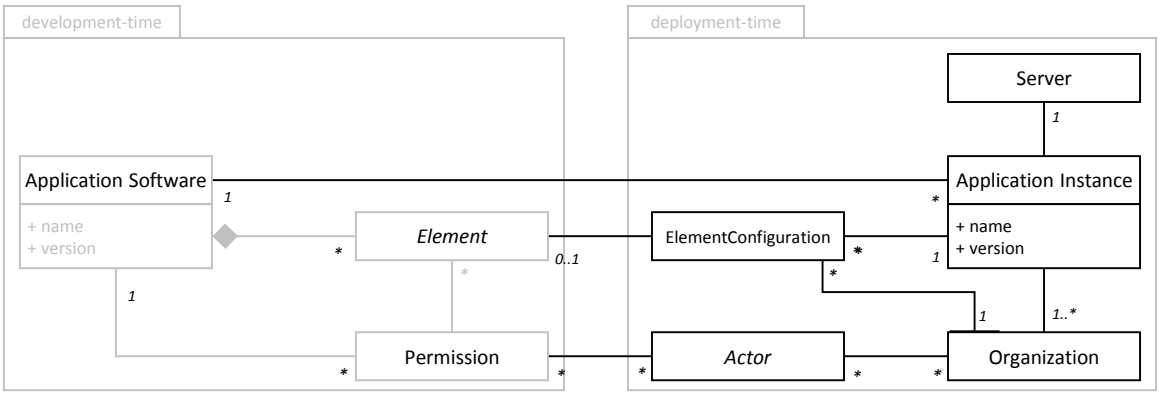


Figure 0-3. Application Instance at deployment-time

The distinction between Application Software and Application Instance is especially difficult when the Software has been developed specifically for one purpose, i.e. for one Instance, which is also the situation where the Configuration is least important and sometimes non-existent. Even in the case of third-party Software, when only one Instance exists in a given organization, the two are often considered one whole.

Run-time: Execution of end-user Application Sessions

At run-time, a User working with an Application Instance establishes a Session. A User (i.e. single individual) typically acts on behalf of or at least as a member of one or several groups.

A Session filters available Elements according to Permissions, and can provide a final, typically shallow layer of Preferences.

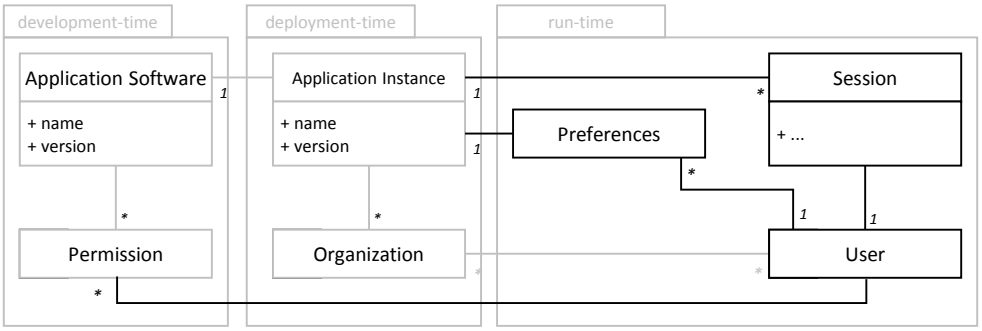


Figure 0-4. Application Session at run-time

Summary

In the rest of this document, we will use the terms Application Software, Element, Application Instance, and Session as described here, and the term Application only when the specifics are not important.

It is worth noting that an Application as seen by the end-user can be considered a "core" with a number of successive adaptation layers. In our example, the first layer is the Configuration at deployment time, the second layer is the Preferences mechanism at run-time, but more layers are possible. The intent of these layers is to gradually fit a generic solution to a specific problem. All commercial-off-the-shelf applications we have encountered implement some variant of this

approach. The boundaries between development, customization and configuration are not always clear.

B. Difficulty of Measuring Fragmentation

When the information system in place provides poor or no support for an important function under a certain group/individual's responsibility, a wide spectrum of reactions can be observed, two extremes of which are listed below. Both demonstrate the impossibility to measure the problem accurately.

Group/individual does the job without shadow application support

In this scenario, since no application performs the function, individuals do. The resulting growth of groups and importance of people skills are major factors of recognition in any corporation. Due to fear to lose their status or job, there is little or no incentive for groups or individuals to recognize the shortcomings of the official applications, or the need for a shadow application.

Group/individual introduces a shadow application

In this opposite scenario, the group or individual has introduced a shadow application to perform the function. The owners are unlikely to volunteer this information, due to the recognition factor mentioned previously plus the fact that some possible outcomes of advertising the application are unlikely to benefit the owner.

- Owner is forced to stop using the shadow application and use the official referential applications
- Responsibility of the shadow application is transferred to another group, officially “so that owner can focus on his core business”, but as a side-effect reducing his agility
- Wider adoption is decided, i.e. the system is promoted to a (more) official status, and thus inherits the associated lack of agility

C. Resolving Inconsistency: Merging

Resolving an inconsistency means *merging* fragments, which implies both the evolution of the schema and the migration of data. We can distinguish two main situations when merging fragments A and B.

- In the simplest case, one fragment (A) is chosen as the convergence target. This implies that the owner of B gives up ownership of his fragment, and trust A's owner to manage subsequent evolutions of the fragment. Executing the convergence thus involves deleting B and migrating the data from B to A.
- In a more complex situation, the agreement is only partial, and a new fragment C must be introduced with only the attributes agreed upon. This can be considered a case of fragment "promotion" to "higher" level perspective. Both owners import fragment C, and remove the redundant attributes from A and B. Data migration is identical to the previous case.

In essence, this operation is very similar to the migration of any enterprise system to a newer version. Simple situations could be managed by fragment owners with graphical support. More complex situations may require complex data transformations, which in the case of low volumes can be envisioned by transiting through a spreadsheet and letting the user perform the translation, and in the case of high volumes by environments similar to aforementioned ETL or data mashup tools allowing to graphically build transformations, and only in the worst case require the assistance of a professional programmer.

D. Fragment Composition Sequence Diagrams

Building the actor-perspective graph in a robust manner is not trivial. The following sections will describe the composition logic through UML2 sequence diagrams.

Notation

The figure below reminds the main elements of the notation, most importantly the distinction between synchronous and asynchronous messages (method invocations), and invocation of a sub-diagram. We will not distinguish between local and remote calls.

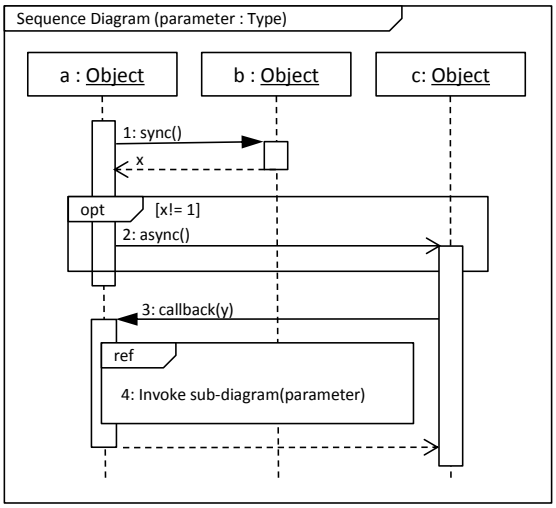


Figure 0-5. UML2 Sequence diagram notation elements

Fragment Composition

The sequence diagrams below illustrate the initialization process of an instance of the Weaver component. The top-level object is the Session which needs a reference to the Directory and the credentials (username) of the current user. The authentication step has been omitted.

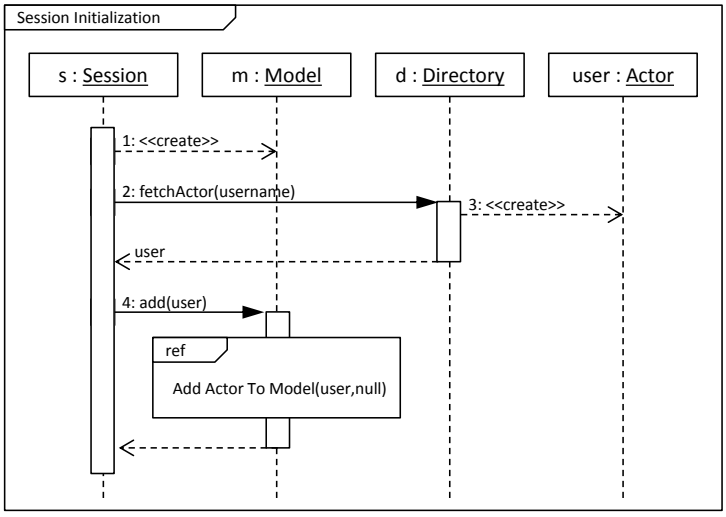


Figure 0-6. Top-level sequence diagram of a Session instance initialization

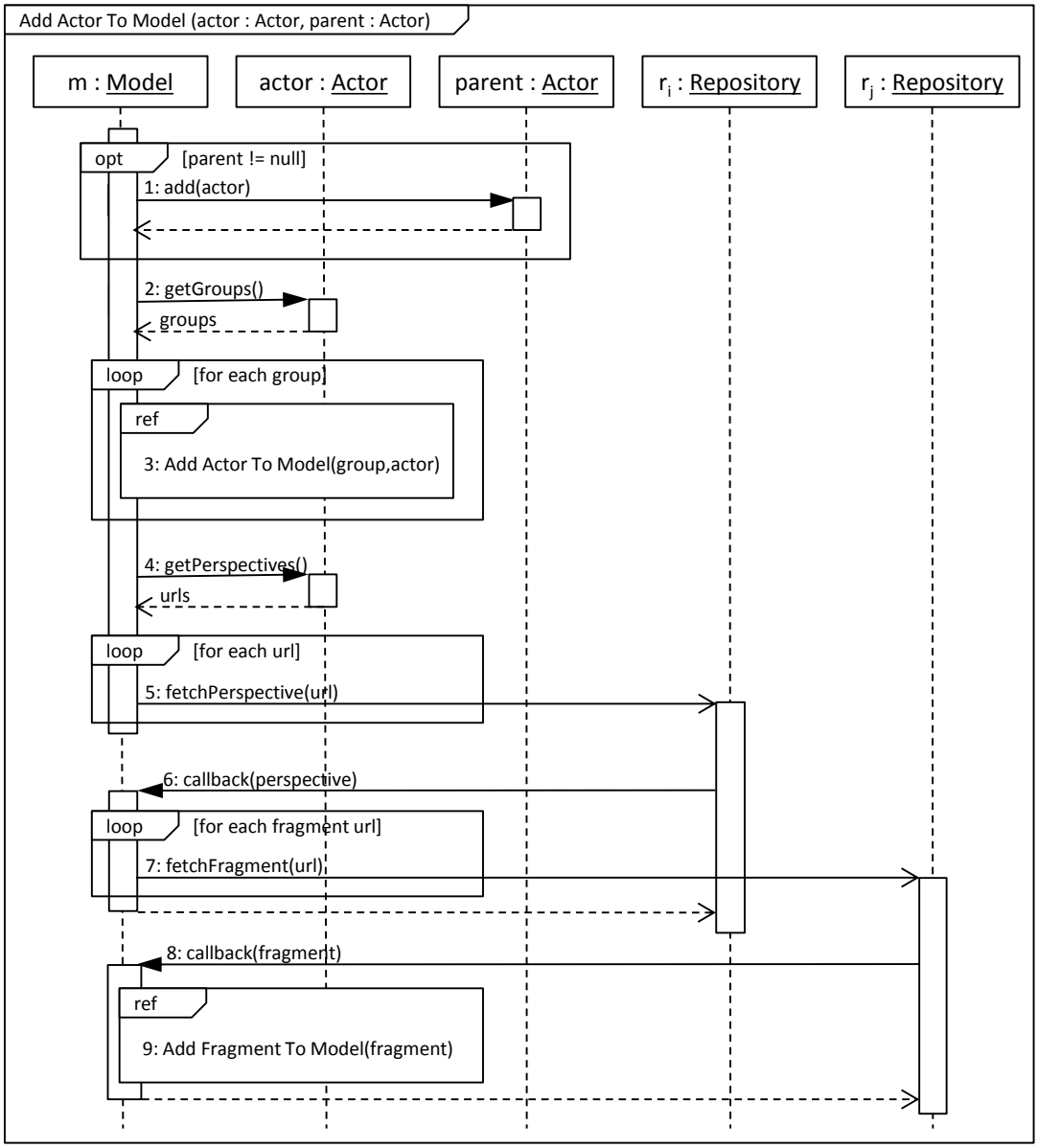


Figure 0-7. Sub-diagram adding an Actor to Model

Several important aspects can be noted in the "Add Actor To Model" diagram.

- The initialization is recursive, and gradually builds the actor graph from the bottom up via message (1).
- Multiple repositories are instantiated on the fly during the Actor graph traversal.
- Both messages (5) and (7) are asynchronous.

The asynchronous nature of the weaver is necessary to meet R6 (resilience). R6 states that unreliable or temporarily missing fragments must not impact the running ones. The asynchronous model composition presented above ensures that all available Fragments get added to the model as soon as possible, even when related fragment requests have not yet been answered.

Adding a Fragment to the Model has been isolated in a separate diagram because it is recursive: a Fragment can reference further Fragments. In the general case, a Fragment which extends another Fragment will fetch its "parent" Fragment. In the particular case of ClassFragments, any attribute

which represents an association with another ClassFragment will need to fetch these "target" Fragments. The figure below only illustrates the general case of fetching the parent Fragment via message (9), and recursively invoking itself in frame (13).

Figure 0-8 also describes the Element composition logic. The asynchronous nature of the weaver cannot guarantee the order in which Fragments become available. The Model thus maintains a heap of "orphan" extension Fragments (8) until the root Fragment is received. Once the root Fragment is available, an Element is instantiated (3) and the orphan extensions retrieved from the heap (4) and added (5). Subsequent extension Fragments skip the heap stage and are added directly to the Element corresponding to their root Fragment (10).

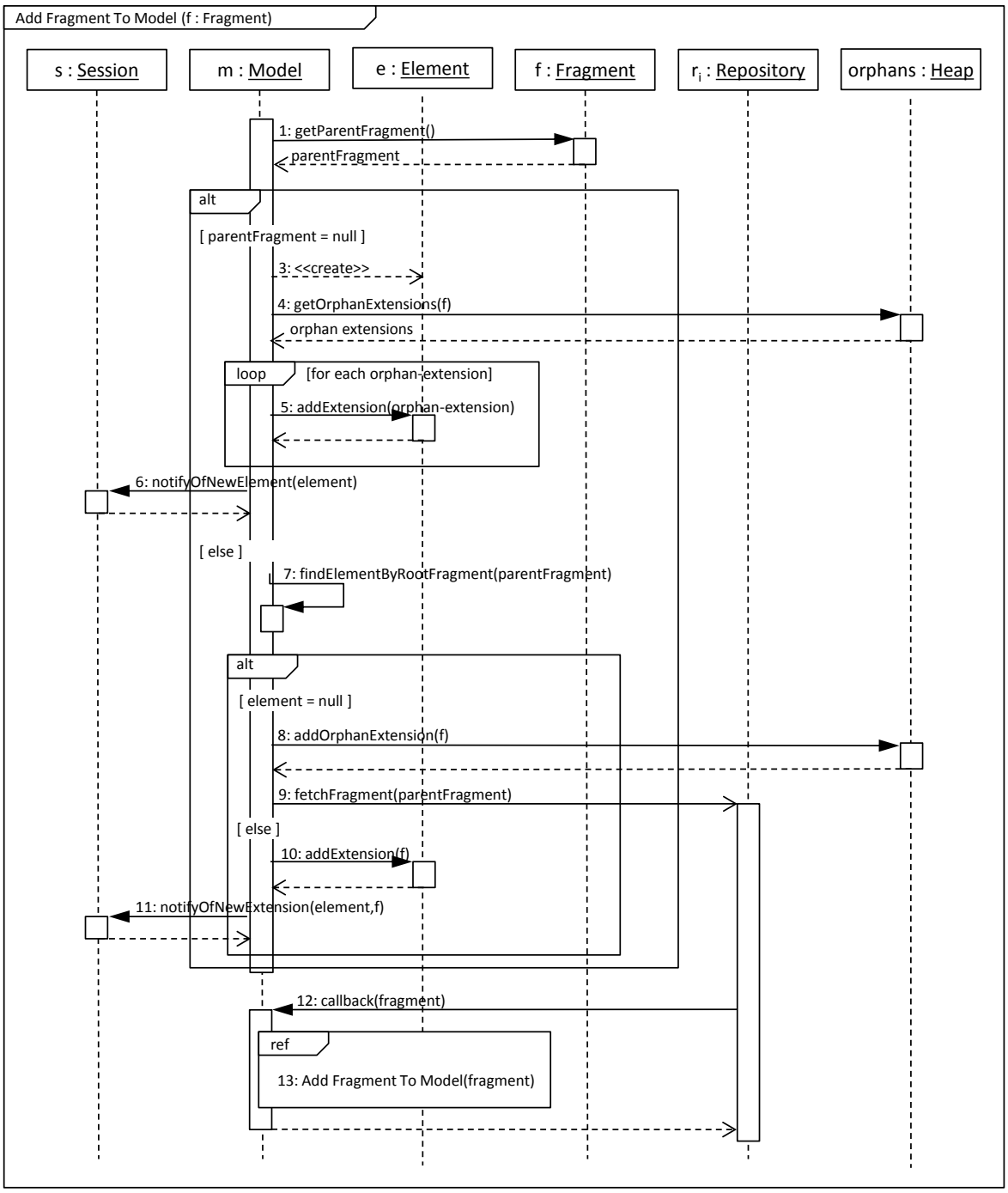


Figure 0-8. Sub-diagram adding a Fragment to Model

Messages (6) and (11) illustrate that the Session is notified of newly available Elements and extensions.

E. User Interface Construction Mechanism

1. Standard Form and Navigation Pattern

The browser provides a standard pattern of forms, allowing all common end-user manipulations (CRUD). The diagram below shows the flow between the 5 main forms of this standard pattern, representing the manipulation of instances of a given ClassElement x.

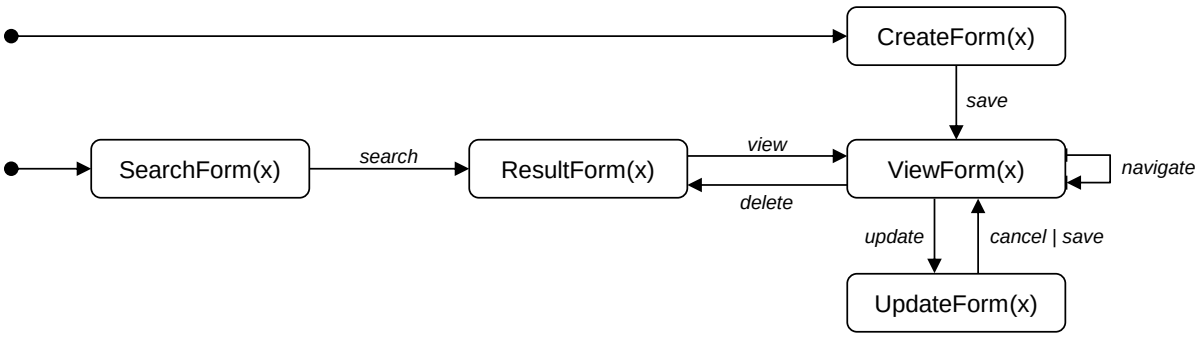


Figure 0-9. Standard form pattern for instance manipulation for a ClassElement x

The standard pattern provides the following forms for each ClassElement.

- A "*search*" form, showing a list of filters on the ClassElements' various attributes, and a single search button. Other modes are possible, often called "Advanced Search", allowing to specify multiple values, ranges, or even text-based query expressions with logical operators.
- A "*result*" form, displaying the list of Instances matching the criteria from the search form, typically providing sort operations and sometimes a bulk update mode. It allows to navigate to the "view" form for each Instance.
- "*view*" form, displaying all attributes of a single Instance. It has an "update" button to switch from read-only widgets to read-write widgets allowing to update the attributes, and a "delete" button to delete the Instance from persistent storage.
- "*update*" form is the same as the "view" form, but allows to change all attributes, and carries the standard "cancel" and "save" operations.
- "*create*" form is similar to the update form, except that it allows to create a new Instance as opposed to updating an existing one.

2. Example Instance Diagram

The diagram below shows the full instance diagram for the example in Figure 6-20.

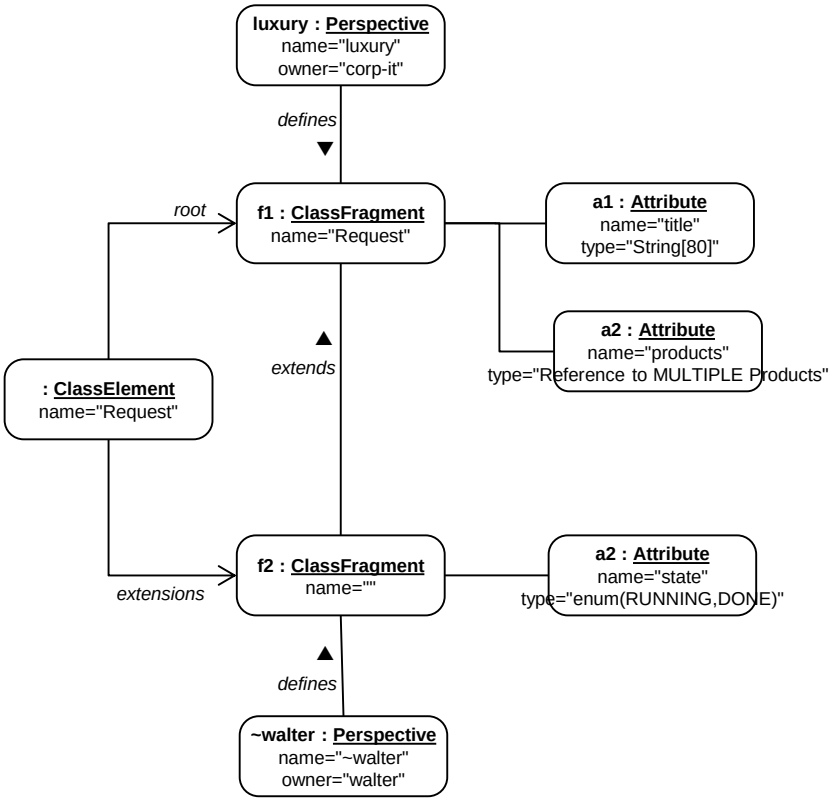


Figure 0-10. Full instance diagram of Weaver classes

3. Classes underlying the user interface construction mechanism

The standard form pattern is implemented by the classes presented in the figure below, which get instantiated when a user action requires them.

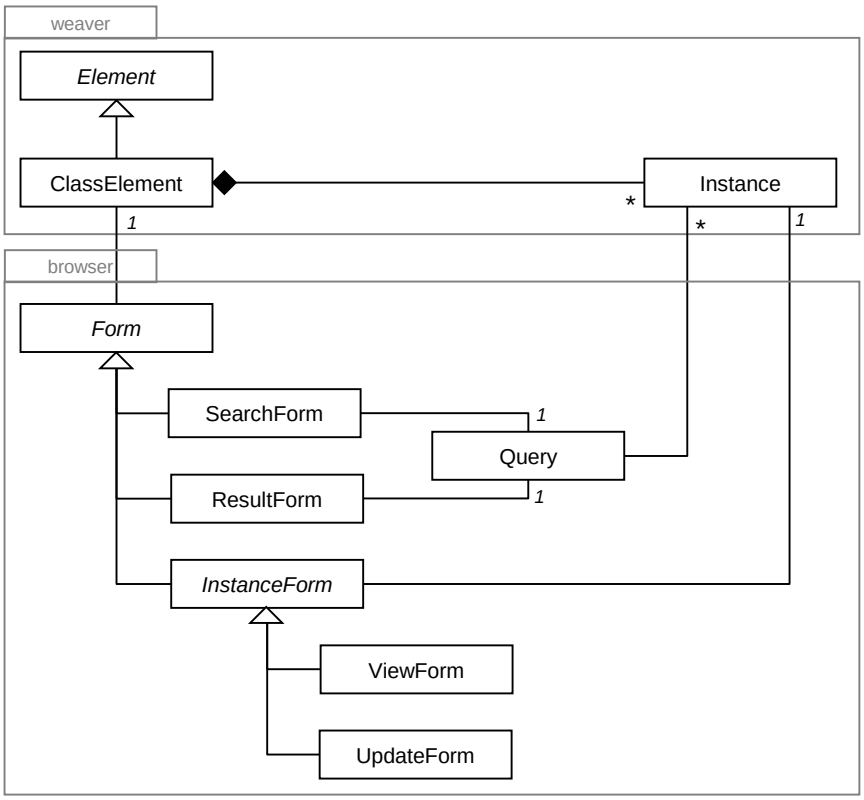


Figure 0-11. Class diagram implementing the standard form pattern

F. Role-Play Experiment Description

1. Protocol

Submit groups of 3 participants to the following role game.

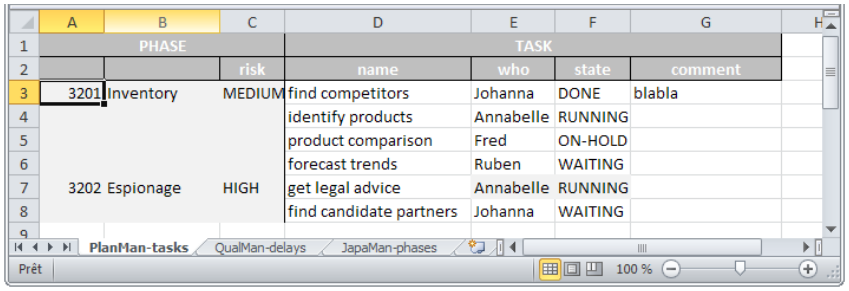
- **Experiment introduction** (0:05)
- **Introduction to use case** (0:05)
  Manufacturing company which introduces a new project tracking application
- **Prototype demonstration** (0:05)
  Demonstrate concept extension and creation, on Conference-Article use-case
- **Mini-training** (0:15)
  Let participants manipulate the Conference-Article example
  Ensure people understand the concepts AND how to manipulate the user interface
  Goal is that UI misunderstandings don't pollute next exercise
- **Assign one role per participant** (0:05)
  Distribute role sheets, describing a few business tasks to perform which require Adaptations and extensions of central app
  Each role sheet requires the participant to :
  a. import[1] and unimport[2] concepts from central application
  b. extend[3] concepts from central app
  c. add[4] concepts in private space
- **Ask participants to perform the business tasks on the role sheet** (30:00)
  Assistance is acceptable to circumvent usability issues of present UI
- **Collect feedback on the experiment** (15:00)
  Self-assessment of concepts understood (0..5) and properly leveraged (0..5)
- **Measurement**
  For each participant, count mandatory tasks accomplished (0..3)

2. Example Role Sheet

Planning Manager

- ## Your job

  You are the planning manager of a department of 80 engineers.
  You track everything at phase-level and even below. It is important for you to know how much risk is associated with each phase. You also need to track *tasks* with a finer granularity than phases. You want to know who is responsible for each task, and what state it is in. So far you have achieved this with the following spreadsheet.

- 

	A	B	C	D	E	F	G
1	PHASE			TASK			
2			risk	name	who	state	comment
3	3201	Inventory	MEDIUM	find competitors	Johanna	DONE	blabla
4				identify products	Annabelle	RUNNING	
5				product comparison	Fred	ON-HOLD	
6				forecast trends	Ruben	WAITING	
7	3202	Espionage	HIGH	get legal advice	Annabelle	RUNNING	
8				find candidate partners	Johanna	WAITING	

  You need to attach tasks to the corresponding phase.

- ## Your main goal

  You want to start the day with the list of running tasks not yet completed

- ## Optional goal

  You are also interested in keeping track of how much effort was spent on the project phases.
  Ideally, you would like to be able to build a bar-chart with total effort per project and per phase.

The other two roles are **Quality** Manager, and **Region Japan** manager.

3. Results

		Session	1			2		
		Role	Planning Manager	Quality Manager	Region Japan	Planning Manager	Quality Manager	Region Japan
		Age	36	30	45	39	49	54
		Background (Academic / Industry)	A	A	A	A	A	A
		Professional developer (Y/N)	Y	Y	Y	N	N	N
		Years of business application usage	6	5	0	10	23	5
		Years of business application development	4	0	5	10	8	5
Number of shadow applications you have…		*… seen*	5	5	0		*"a lot"*	20
		… used	3	3	0		*"a lot"*	20
		… developed	1	0	0	3	*"a lot"*	2
		Has imported	***1***	***1***	***1***	***1***	***1***	***1***
		Has extended	***1***	***1***	***1***	***1***	***1***	***1***
		Has created	***1***	***1***	***1***	***1***	***1***	***1***
		Has unimported	***1***	***1***	***0***	***1***	***1***	***1***
		Successfully done job	***1***	***1***	***1***	***1***	***1***	***1***

		Session	3			4		
		Role	Planning Manager	Quality Manager	Region Japan	Planning Manager	Quality Manager	Region Japan
		Age	45	47	48	32	30	27
		Background (Academic / Industry)	I	I	I	I	I	I
		Professional developer (Y/N)	N	N	N	Y	Y	Y
		Years of business application usage	20	26	22	8	7	3
		Years of business application development	5	0	0	8	7	3
Number of shadow applications you have…		*… seen*	>50	20	60+	>100	16	5
		… used	~20	5	30+	>50	10	3
		… developed	10	1	12	15	5	2
		Has imported	***1***	***1***	***1***	***1***	***1***	***1***
		Has extended	***1***	***1***	***1***	***1***	***1***	***1***
		Has created	***1***	***1***	***1***	***1***	***1***	***1***
		Has unimported	***1***	***0***	***1***	***1***	***1***	***1***
		Successfully done job	***1***	***1***	***1***	***1***	***1***	***1***

Bibliography

[1] Object Management Group, "UML 2.0," [Online]. Available: http://www.omg.org/spec/UML/2.0/. [Accessed June 2012].

[2] M. Van Oosterhout, E. Waarts and J. Van Hillegersberg, "Assessing Business Agility: A Multi-Industry Study in the Netherlands," *Business agility and information technology,* vol. 180, 2005.

[3] M. Strohmaier and H. Rollett, "Future Research Challenges in Business Agility – Time, Control and Information Systems," in *Seventh IEEE International Conference on E-Commerce Technology*, 2005.

[4] L. Mathiassen and J. Pries-Heje, "Business Agility and Diffusion of Information Technology," *European Journal of Information Systems,* no. 15, 2006.

[5] S. Goldman, R. Nagel and K. Preis, Agile Competitors and Virtual Organizations, Van Nostrand Reinhold, 1995.

[6] H. Sharifi and Z. Zhang, "A methodology for achieving agility in manufacturing organizations: An introduction," *International Journal of Production Economics,* no. 62, pp. 7-22, 1999.

[7] R. Dove, Response Ability: The Language, Structure and Culture of the Agile Enterprise, Wiley, 2001.

[8] M. J. Hooper, D. Steeple and C. Winters, "Costing Customer Value: An approach for the agile enterprise," *International Journal of Operations and Production Management,* no. 21, pp. 630-644, 2001.

[9] K. Conboy and B. Fitzgerald, "Toward a Conceptual Framework of Agile Methods:A Study of Agility in Different Disciplines," in *ACM Workshop on Interdisciplinary Software Engineering Research*, 2004.

[10] S. Wadhwa and K. Rao, "Flexibility and agility for enterprise synchronization: Knowledge and innovation management towards flexagility," *Studies in Informatics and Control,* vol. 12, no. 2, pp. 111-128, 2003.

[11] M. Van Oosterhout, E. Waarts, E. Van Heck and J. Van Hillegersberg, "Business Agility: Need, Readiness and Alignment with IT strategies," in *Agile Information Systems*, Elsevier, 2007, pp. 52-69.

[12] T. Margaria and B. Steffen, "Continuous Model-Driven Engineering," *IEEE Computer,* vol. 42, no. 10, pp. 106-109, 2009.

[13] J. A. Carvalho, "Information System? Which One Do You Mean?," in *Information Systems Concepts: An Integrated Discipline Emerging*, Kluwer Academic Publishers, 2000, pp. 259-280.

[14] M. Quast and J.-M. Favre, "Towards Social Information Systems," in *3rd International Workshop on Social Software Engineering*, 2010.

[15] K. Lindblad-Gidlund, "When and How Do We Become a "User" ?," in *Reframing Humans in Information Systems Development*, 2011, pp. 211-225.

[16] M. Spahn and V. Wulf, "End-User Development for Individualized Information Management: Analysis of Problem Domains and Solution Approaches," in *International Conference on Enterprise Information Systems (ICEIS)*, 2009.

[17] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* 1970.

[18] M. M. Zloof, "A language for office and business automation," in *Data Base Design Techniques: Lecture Notes in Computer Science*, 1982.

[19] J. Sametinger, "On a Taxonomy for Software Components," in *International Workshop on Component-Oriented Programming (WCOP)*, 1996.

[20] B. Boehm, "A View of 20th and 21st Century Software Engineering," in *International Conference on Software Engineering (ICSE)*, 2006.

[21] J. Bonar, N. Lehrer, K. Looney, R. Schnier, J. Russel, T. Selker and S. Nickolas, "Components on the Internet (panel)," in *International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, 1996.

[22] L. Ellison, Writer, *Oracle World 2011 keynote (http://www.youtube.com/watch?v=XLmtD3CanpM).* [Performance]. The Oracle Corporation, 2011.

[23] M. Davidsen and J. Krogstie, "Information System Evolution over the Last 15 Years," in *International Conference on Advanced Information System Engineering (CAiSE)*, 2010.

[24] M. L. Markus and C. Tanis, "The Enterprise Systems Experience - From Adoption to Success," in *Framing the Domains of IT Research: Glimpsing the Future Through the Past*, Pinnaflex Educational Resources, 200, pp. 173-207.

[25] C. Soh, S. D. Kien and J. Tay-Yap, "Cultural Fits and Misfits: Is ERP a Universal Solution?," *Communications of the ACM,* vol. 43, no. 4, pp. 47-51, 2000.

[26] L. Brehm, A. Heinzl and M. L. Markus, "Tailoring ERP Systems: A Spectrum of Choices and their Implications," in *International Conference on System Sciences*, 2001.

[27] T. H. Davenport, "Putting the Enterprise into the Enterprise System," *Harvard Business Review,* vol. 76, no. 4, pp. 121-131, 1998.

[28] A. Gurram, B. Mo and R. Gueldemeister, "A Web Based Mashup Platform for Enterprise 2.0," in *International Conference on Web Information System Engineering (WISE)*, 2008.

[29] V. Hoyer and K. Stanoevska-Slabena, "The Changing Role of IT Departments in Enterprise Mashup Environments," in *Service-Oriented Computing*, Springer-Verlag, 2008, pp. 148-154.

[30] S. Newell, E. Wagner and G. David, "Clumsy Information Systems : A Critical Review of Enterprise Systems," in *Agile Information Systems*, Elsevier, 2007, pp. 163-177.

[31] M. S. Ackerman, "The Intellectual Challenge of CSCW: The Gap Between Social Requirements and Technical Feasibility," *Human-Computer Interaction,* vol. 15, no. 2, pp. 179-203, 2000.

[32] J. Luftman and R. Kempaiah, "An Update on Business-IT Alignment: "A Line" Has Been Drawn," *MIS Quarterly Executive,* vol. 6, no. 3, pp. 165-177, 2007.

[33] S. Bitzer and M. Schumann, "Mashups: An Approach to Overcoming the Business/IT Gap in Service-Oriented Architectures," *Lecture Notes in Business Information Processing,* 2009.

[34] S. Clarke, W. Harrison, H. Ossher and P. Tarr, "Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code," in *International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, 1999.

[35] T. Nestler, "Towards a Mashup-driven End-User Programming of SOA-based Applications," in *Internationel Conference on Information Integration and Web-based Applications and Services (iiWAS)*, 2008.

[36] R. Zarnekow, W. Brenner and U. Pilgram, Integrated Information Management: Applying Successful Industrial Concepts in IT, Springer, 2006.

[37] A. Benz, Einleitung: Governance - Modebegriff oder nützliches sozialwissenschaftliches Konzept, 2004.

[38] P. Webb, C. Pollard and G. Ridley, "Attempting to Define IT Governance: Wisdom or Folly?," in *International Conference on System Sciences*, 2006.

[39] M. Handel and S. Poltrock, "Working Around Official Applications," in *Computer-Supported Collaborative Work (CSCW)*, 2011.

[40] B. A. Nardi and J. R. Miller, "An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development," in *Computer-Supported Collaborative Work (CSCW)*, 1990.

[41] D. McIlroy, "Mass-Produced Software Components," in *Software Engineering, Report on a conference sponsored by the NATO Science Committee*, 1969.

[42] Gotts, "A New Cloud: The Stealth Cloud?," 2010. [Online]. Available: http://www.cio.com/article/630164. [Accessed June 2012].

[43] W. Hordijk and R. Wieringa, "Rationality of Cross-System Data Duplication: A Case Study," in *International Conference on Advanced Information System Engineering (CAiSE)*, 2010.

[44] W. Harrisson, "The Dangers of End-User Programming," *IEEE Software,* vol. 21, no. 4, pp. 5-7, 2004.

[45] L. Ellram, "Total Cost of Ownership: Elements and Implementation," *Journal of Supply Chain Management,* vol. 29, no. 4, pp. 2-11, 1993.

[46] S. Lohmann, S. Dietzold, P. Heim and H. N., "A Web Platform for Social Requirements Engineering," in *Software Engineering*, 2009.

[47] R. Kling, "Cooperation, Coordination and Control in Computer-Supported Work," *Communications of the ACM,* vol. 34, no. 12, pp. 83-88, 1991.

[48] T. Finholt and L. S. Sproull, "Electronic Groups at Work," *Organizational Science,* vol. 1, 1990.

[49] A. Van Lansweerde and E. Letier, "Integrating Obstacles in Goal-Driven Requirements Engineering," in *International Conference on Software Engineering (ICSE)*, 1998.

[50] W. N. Robinson, "Negotiation Behaviour During Multiple Agent Specification: A Need for Automated Conflict Resolution," in *International Conference on Software Engineering (ICSE)*, 1990.

[51] P. J. Denning, C. Gunderson and H.-R. R., "Evolutionary System Development," *Communications of the ACM,* vol. 51, no. 12, pp. 29-31, 2008.

[52] J. Kitzinger, "Focus Groups," in *Qualitative Research in Health Care*, Blackwell Publishing Ltd, 2007.

[53] P. Thibodeau, "IBM on path to cut internal apps by 85%," April 2012. [Online]. Available: http://www.computerworld.com/s/article/9226430/IBM_on_path_to_cut_internal_apps_by_85_. [Accessed June 2012].

[54] G. Fischer and A. Girgensohn, "End-User Modifiability in Design Environments," in *Proceedings Conference on Human Factors in Computing Systems (CHI'90)*, 1990.

[55] A. MacLean, K. Carter, L. Lovstrand and T. Moran, "User-Tailorable Systems: Pressing the Issues with Buttons," in *Proceedings of Human Factors in Computing Systems (CHI'90)*, 1990.

[56] A. Mørch, G. Stevens, M. Won, Klann, M., Y. Dittrich and V. Wulf, "Component-Based Technologies for End-User Development," *Communications of the ACM,* vol. 47, no. 9, p. 59–62, 2004.

[57] R. Wieringa, "An Introduction to Requirements Traceability," Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1995.

[58] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee and R. Stafford, Patterns of Enterprise Application Architecture, Addisson-Wesley, 2003.

[59] G. J. Myers and L. L. Constantine, "Structured design," *IBM Systems Journal,* 1979.

[60] K. C. Desouza, Agile Information Systems: Conceptualization, Construction and Management, Elsevier, 2009.

[61] B. Light, C. P. Holland and W. K., "ERP and best of breed: a comparative analysis," *Business Process Management Journal,* vol. 7, no. 3, pp. 216-224, 2001.

[62] R. Orfali, D. Harkey and J. Edwards, The essential distributed objects survival guide, John Wiley & Sons, 1995.

[63] E. F. Codd, "Extending the database relational model to capture more meaning," *ACM Transactions on Database Systems (TODS),* 1979.

[64] T. Leveque, "Définition et contrôle des politiques d'évolution dans les projets logiciels (doctoral dissertation)," Laboratoire Informatique de Grenoble (LIG), 2010.

[65] Y. V. Natis, "Service-Oriented Architecture Scenario," Gartner Research, 2003.

[66] N. Josuttis, SOA in practice: The Art of Distributed System Design, Sebastopol: O'Reilly Media Inc, 2007.

[67] K. Bennett, D. Budgen, P. Brereton, L. Macaulay and M. Munro, "Service-Based Software: The Future for Flexible Software," in *Asia-Pacific Software Engineering Conference*, 2000.

[68] A. Michlmayer, F. Rosenberg, C. Platzer, M. Treiber and S. Dustdar, "Towards recovering the broken SOA triangle : a software engineering perspective," in *International Workshop on Service-Oriented Software Engineering*, New York, 2007.

[69] X. Liu, Y. Hui, W. Sun and H. Liang, "Towards service composition based on mashups," in *IEEE International Conference on Service Computing*.

[70] T. Janner, V. Canas, J. Hierro, D. Licano, M. Reyers, C. Schroth and J. Soriano, "Enterprise Mashups: Putting a face on next generation global SOA," in *International Conference on Web Information Systems Engineering (WISE)*, 2007.

[71] J. Soriano, D. Lizcano, M. Canas, M. Reyes and J. Hierro, "Foster Innovation in a Mashup-oriented Enterprise 2.0 Collaboration Environment," *System and Information Sciences Notes,* vol. 1, no. 1.

[72] S. Watt, "Mashups - the evolution of the SOA - part 2 : Situational applications and the mashup ecosystem," 2007. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-soa-mashups2.

[73] V. Hoyer and M. Fischer, "Market Overview of Enterprise Mashup Tools," in *International Conference on Service-Oriented Computing (ICSOC)*, 2008.

[74] T. Erl, Service-Oriented Architecture: Concepts, Technology & Design, Prentice Hall, 2005.

[75] M. Cañas, J. Hierro, V. Hoyer, T. Janner, D. Lizcano, M. Reyes and C. Schroth, "Enterprise Mashups: Putting a face on the next generation global SOA," in *International Conference on Web Information Systems Engineering (WISE)*, 2007.

[76] S. Peenikal, "Mashups and the enterprise," Mphasis, a Hewlett-Packard Company, 2009.

[77] E. Cerami, Web Services Essentials, O'Reilly & Associates, Inc, 2002.

[78] L. Santillo, "Seizing and Sizing SOA Applications with COSMIC Function Points," in *Software Measurement European Forum*, 2007.

[79] R. Thiagarajan, A. K. Srivastava, A. K. Pujari and B. K. Visweswar, "BPML : a process modeling language for dynamic business models," in *Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2002.

[80] Yahoo!, "Yahoo! Pipes," [Online]. Available: pipes.yahoo.com. [Accessed August 2012].

[81] K. Scarfone, W. Jansen and M. Tracy, "Guide to General Server Security," National Institute of Standards and Technology, 2011.

[82] W. Van Der Aalst, "Web Service Composition Languages: Old Wine in New Bottles?," in *Euromicro Conference*, 2003.

[83] M. Abu Jarour, F. Naumann and M. Craculeac, "Collecting, Annotating and Classifying Public Web Services," in *On the move to meaningful internet systems*, 2010.

[84] A. Finkelstein, J. Kramer and M. Goedicke, "ViewPoint Oriented Software Development," in *International Workshop on Software Engineering and its Applications*, 1990.

[85] M. Sabetzadeh, A. Finkelstein and M. Goedicke, "ViewPoints," in *Encyclopedia of Software Engineering*, P. Laplante, 2010.

[86] M. Lubars, C. Potts and C. Richter, "A review of the state of the practice in requirements modeling," in *IEEE International Symposium on Requirements Engineering*, 1993.

[87] P. Loucopoulos, "Requirements Engineering: Panacea or Predicament? (keynote)," in *International Conference on Enterprise Information Systems (ICEIS)*, 2012.

[88] D. C. Schmidt, "Model-Driven Engineering," *IEEE Computing,* vol. 39, 2006.

[89] J.-M. Favre, J. Estublier and M. Blay-Fornarino, L'ingénierie dirigée par les modèles: au-delà du MDA, Hermès - Lavoisier, 2006.

[90] D. S. Hovorka and M. Germonprez, "Reflection, Tinkering, and Tailoring: Implications for Theories of Information System Design," in *Reframing Humans in Information Systems Development*, Springer-Verlag, 2011, pp. 135-149.

[91] "Evolutionary Application Development," in *Reframing Humans in Information Systems Development*, Springer-Verlag, 2011, pp. 151-171.

[92] S. Soi and M. Baez, "Domain-specific Mashups: From All to All You Need," in *International Conference on Web Engineering*, 2010.

[93] N. Ahmadi, M. Jazayeri, F. Lelli and A. Repenning, "Towards the Web Of Applications : Incorporating End User Programming into the Web 2.0 Communities," in *Proceedings of the 2nd international workshop on Social software engineering and applications (SoSEA)*, 2009.

[94] W. Harrisson and H. Ossher, "Subject-Oriented Programming (A Critique of Pure Objects)," in *International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, 1993.

[95] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier and J. Irwin, "Aspect-Oriented Programming," in *European Conference on Object-Oriented Programming (ECOOP)*, 1997.

[96] The Oracle Corporation, "An Introduction to the Java EE 5 Platform," May 2006. [Online]. Available: http://java.sun.com/developer/technicalArticles/J2EE/intro_ee5/. [Accessed July 2012].

[97] Standish Group International, "Chaos," 1994.

[98] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley , 1999.

[99] J. V. Sutherland and K. Schwaber, "The SCRUM Methodology," in *Business object design and implementation: OOPSLA workshop*, 1995.

[100] "The Agile Manifesto," [Online]. Available: http://agilemanifesto.org/. [Accessed June 2012].

[101] G. Lee and W. Xia, "Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility," *MIS Quarterly,* vol. 34, no. 1, pp. 87-114, 2010.

[102] G. Fitzgerald, "Achieving Flexble Information Systems: The Case for Improved Analysis," *Journal of Information Technology,* vol. 5, no. 1, pp. 5-11, 2001.

[103] P. Robertson, "Integrating Legacy Applications with Modern Corporate Applications," *Communications of the ACM,* vol. 40, no. 5, pp. 39-46, 1997.

[104] D. Stamoupolis, P. Kanellis and D. Martakos, "Tailorable Information Systems: Resolving the Deadlock of Changing User Requirements," *Journal of Applied System Studies,* vol. 2, no. 2, 2001.

[105] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, 2009.

[106] S. Jansen, G.-J. Houben and S. Brinkkemper, "Customization Realization in Multi-tenant Web Applications: Case Studies from the Library Sector," in *International Conference on Web Engineering (ICWE)*, 2010.

[107] D. S. Linthicum, Enterprise Application Integration, Addisson-Wesley, 2000.

[108] F. Menge, "Enterprise Service Bus," in *Free and Open Source Software Conference*, 20007.

[109] A. Laftsidis, "Enterprise Application Integration," IBM, 2000.

[110] A. Umar and A. Zordan, "Enterprise Ontologies for Planning and Integration of Business: A Pragmatic Approach," *IEEE Transactions on Engineering Management,* vol. 56, no. 2, pp. 352-371 , 2009.

[111] P. Vassiliadis, "A Survey of Extract–Transform–Load Technology," *International Journal of Data Warehousing & Mining,* vol. 5, no. 3, pp. 1-27, 2009.

[112] A. Y. Halevy, H. Ashish, D. Bitton, M. Carey, D. Draper, J. Pollock, A. Rosenthal and V. Sikka, "Enterprise information integration: Successes, Challenges and Controversies," in *ACM SIGMOD International Conference on Management of Data*, 2005.

[113] "Merriam-Webster Online Dictionary," [Online]. Available: http://www.merriam-webster.com/.

[114] Project Management Institute, [Online]. Available: www.pmi.org.

[115] S. Star, "The Structure of Ill-Structured Solutions : Boundary Objects and Heterogeneous Problem Solving," in *Distributed artificial intelligence*, 1990.

[116] C. Kimble, C. Grenier and K. Goglio-Primard, "Innovation and Knowledge sharing across professional boundaries: Political interplay between boundary objects and brokers," *International Journal of Information Management,* vol. 30, pp. 437-444, 2010.

[117] R. Hyde, "The Fallacy of Premature Optimization," *Ubiquity,* 2009.

[118] N. Balasubramaniam, "User-Generated Content," in *Business Aspects of the Internet of Things, Seminar of Advanced Topics*, 2008.

[119] J. Surowiecki, The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business,Economies, Societies and Nations, Doubleday Books, 2004.

[120] A. Namoun, T. Nestler and A. De Angeli, "End User Requirements for the Composable Web," in *International Conference on Web Engineering (ICWE)*, 2010.

[121] D. Tapscott and A. D. Williams, Wikinomics: How Mass Collaboration Changes Everything, Portfolio, 2006.

[122] A. Hotho, R. Jäschke, C. Schmitz and G. Stumme, "Information Retrieval in Folksonomies : Search and Ranking," in *Extended Semantic Web Conference*, 2006.

[123] K. Aberer, P. Cudré-Mauroux, A. M. Ouksel, T. Catarci, M.-S. Hacid, A. Illarramendi, V. Kashyap, M. Mecella, E. MenaetErich and J. Neuhold, "Emergent Semantics: Principles and Issues," in *Database Systems for Advanced Applications*, 2004.

[124] R. Jaschke, L. Marinho, A. Hotho, L. Schmidt-Thieme and G. Stumme, "Tag Recommendations in Folksonomies," in *11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2007.

[125] J. A. Konstan and J. Riedl, "Recommender Systems: From Algorithms to User Experience," *User Modeling and User-Adapted Interaction,* vol. 22, pp. 101-123, 2012.

[126] G. Adomavicius and A. Tuzhilin, "Towards the Next Generation of Recommender Systems:A Survey of the State-of-the-Art and Possible Extensions," *IEEE Transactions on Knowledge and Data Engineering,* 2005.

[127] M. Balabanovic and Y. Shoham, "Fab: Content-based, collaborative recommendation," *Communications of the ACM,,* vol. 40, no. 3, pp. 66-72, 1997.

[128] X. Zhou, Y. Xu, Y. Li, A. Josang and C. Cox, "The state-of-the-art in personalized recommender systems for social networking," *Artificial Intelligence Review,* vol. 37, no. 2, pp. 119-132, 2012.

[129] T. Babaian and W. Lucas, "Leveraging Usage History To Support Enterprise System Users," in *International Conference on Enterprise Information Systems (ICEIS)*, 2012.

[130] M. W. Van Alstyne and E. Brynjolfsson, "Global Village or CyberBalkans: Modeling and Measuring the Integration of Electronic Communities," *Management Science,* vol. 51, no. 6, pp. 851-868, 2005.

[131] D. Fleder and K. Hosanger, "Blockbuster culture's next rise or fall: the impact of recommender systems on sales diversity," *Management Science,* vol. 55, no. 5, pp. 697-712, 2007.

[132] J. Oakland, Statistical Process Control, John Wiley and Sons , 1986.

[133] F. Schwagereit, S. Sizov and S. Staab, "Finding optimal policies for online communities with cosimo," in *Web Science*, 2010.

[134] F. Schwagereit, A. Scherp and S. Staab, "Survey on Governance of User-generated Content," in *Web Science*, 2011.

[135] S. Angeletou, M. Rowe and H. Alani, "Modelling and Analysis of User Behaviour in Online Communities," in *International Conference on the Semantic Web (ISWC)*, 2011.

[136] S. Staab, "Managing Online Business Communities (keynote)," in *International Conference on Enterprise Information Systems (ICEIS)*, 2012.

[137] B. Nuseibeh, J. Kramer and A. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification," *IEEE Transactions on Software Engineering,* vol. 20, no. 10, pp. 760-773 , 1994.

[138] R. Balzer, "Tolerating Inconsistency," in *International Conference on Software Engineering (ICSE)*, 1991.

[139] B. Nuseibeh, S. Easterbrook and A. Russo, "Making Inconsistency Respectable in Software Development," *Journal of Systems and Software,* vol. 56, no. 58, pp. 171-180, 2000.

[140] C. Ghezzi and B. A. Nuseibeh, "Special Issue on Managing Inconsistency in SOftware Development," *Transactions on Software Engineering,* vol. 24, no. 11, pp. 906-1001, 1998.

[141] R. W. Schwanke and G. E. Kaiser, "Living with Inconsistency in Large Systems," in *International Workshop on Software Version and Configuration Control*, 1988.

[142] A. Kittur and R. E. Kraut, "Harnessing the Wisdom of Crowds in Wikipedia: QualityThrough Coordination," in *Computer-Supported Collaborative Work (CSCW)*, 2008.

[143] B. Nuseibeh, "Ariane 5: Who Dunnit?," *IEEE Software,* vol. 14, no. 3, pp. 15-16, 1997.

[144] K. Zeilenga, "RFC 4510 - Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map," Internet Engineering Task Force (IETF), 2006.

[145] C. Szyperski, Component Software: Beyond Object-Oriented Programming (2nd Edition), Addison-Wesley Professional, 2002.

[146] S. B. Navathe, K. Karlapalem and M. Ra, "A mixed fragmentation methodology for initial distributed database design," *Journal of Computer and Software Engineering,* 1995.

[147] H. Roinestad, J. Burgoon, B. Markines and F. Menczer, " Incentives for social Annotation," in *20th ACM conference on Hypertext and hypermedia*, 2009.

[148] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures (doctoral dissertation)," University of California, Irvine, 2000.

[149] J. Edwards, "Example Centric Programming," in *International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, 2004.

[150] S. R. Vaughn, J. S. Schumm and J. M. Sinagub, Focus Group Interviews in Education and Psychology, Sage Publications, 1996.

[151] J. Estublier and R. Casallas, "Three dimensional versioning," in *Software Configuration Management*, Springer, 1995, pp. 118-135.

[152] J. Gray, "The Transaction Concept: Virtues and Limitations," in *Very Large Databases (VLDB)*, 1981.

[153] B. G. Lindsay and P. Selinger, "Notes on Distributed Databases," in *Distributed Databases*, 1979.

[154] E. Brewer, "Towards Robust Distributed Systems," in *ACM Symposium on Principles of Distributed Computing*, 2000.

[155] R. Kling, "What is Social Informatics and Why Does it Matter?," January 1999. [Online]. Available: http://dlib.org/dlib/january99/kling/01kling.html.

[156] F.-Y. Wang, D. Zeng, K. M. Carley and W. Mao, "Social Computing: From Social Informatics to Social Intelligence," *Intelligent Systems (IEEE),* vol. 22, no. 2, pp. 79-83 , 2007.

[157] I. King, J. L. and K. T. C., "A brief survey of computational approaches in Social Computing," in *International Joint Conference on Neural Networks (IJCNN)*, 2009.

[158] M. Parameswaran and A. B. Whinston, "Research issues in social computing," *Journal of the Association for Information Systems,* vol. 8, no. 6, pp. 336-350, 2007.

[159] M. C. Daconta, L. J. Obrst and K. T. Smith, The Semantic Web : A Guide to the Future of XML, Web Services, and Knowledge Management, 2003.

[160] C. Bizer, T. Heath and T. Berners-Lee, "Linked Data - The Story So Far," *International Journal on Semantic Web and Information Systems,* vol. 5, no. 3, 2009.

[161] T. Berners-Lee, "Linked Data - Design Issues," July 2006. [Online]. Available: http://www.w3.org/DesignIssues/LinkedData.html. [Accessed June 2012].

[162] M. Quast and M. Handel, "Social Information Systems: The End of Shadow Applications?," in *International Conference on Enterprise Information Systems (ICEIS)*, 2012.

[163] G. Thomson, "BYOD: enabling the chaos," *Network Security,* vol. 2012, no. 2, pp. 5-8, 2012.

[164] F. Graham, "BBC News - BYOD: Bring your own device could spell end for work PC," [Online]. Available: http://www.bbc.co.uk/news/business-17017570.

[165] W. Vogels, "Eventually Consistent," *ACM QUEUE,* pp. 14-19, 2008.

[166] G. Fischer, "End-User Development and Meta-design: Foundations for Cultures of Participation," in *2nd International Symposium on End-User Development*, 2009.

[167] J. W. Ross, "Creating a Strategic IT Architecture Competency: Learning in Stages," MIT Sloan School of Management, 2003.

[168] V. Markl, M. Altinel, D. Simmen and A. Singh, "Data Mashups for Situational Applications," in *International Workshop on Model-Based Software and Data Integration (MBSDI)*, 2008.

[169] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns:Elements od Reusable Software, Addison-Wesley, 1994.

[170] S. Garcia, "Ingénierie Concurrente en Génie Logiciel : Céline (doctoral dissertation)," Laboratoire Informatique de Grenoble (LIG), 2006.

[171] J. L. Eveleend and C. Verhoef, "The Rise and Fall of the Chaos Report Figures," *IEEE Software,* pp. 30-36, 2010.

[172] D. R. Mauro and K. J. Schmidt, Essential SNMP, O'Reilly & Associates, 2001.

[173] E. Schenk and C. Guittard, "Towards a characterization of crowdsourcing practices," *Journal of Innovation Economics,* vol. 1, no. 7, 2001.

[174] A. Sutcliffe, "Evaluating the costs and benefits of end-user development," *ACM SIGSOFT Software Engineering Notes,* vol. 30, 2005.

[175] K. Bodker, F. Kensing and J. Simonsen, "Participatory Design in Information Systems Development," in *Reframing Humans in Information Systems Development*, Springer-Verlag, 2011, pp. 115-134.

[176] V. Grover, M. Joong Cheon and J. T. C. Teng, "A descriptive study on the outsourcing of information systems functions," *Information & Management,* vol. 27, no. 1, p. 33–44, 1994.

Index

RESUME

Les systèmes d'information d'entreprise actuels s'articulent autour d'applications centrales lourdes, qui ne fournissent pas l'agilité nécessaire pour survivre dans un environnement économique hautement concurrentiel. De nombreux acteurs (unités commerciales, individus, équipes et communautés) doivent introduire leurs propres applications pour pallier à ces limitations, avec pour résultat un système d'information fragmenté, incohérent et impossible à gouverner.

Cette étude propose un paradigme d'architecture d'entreprise alternatif, qui s'appuie sur une décomposition plus fine du système d'information et une distribution différente des responsabilités. Il permet à tout acteur de contribuer au système d'information en introduisant des fragments, privés ou partagés avec d'autres acteurs, qui peuvent ensuite être composés pour former des applications dédiées à un profil. Les récents mécanismes de l'informatique sociale sont proposés pour gérer les volumes potentiels importants de fragments émergeant de la communauté d'employés.

L'objectif des systèmes d'informations sociaux est à la fois d'améliorer la cohérence et la gouvernabilité du système d'information de l'entreprise et d'exploiter l'intelligence et l'énergie collective de l'entreprise à des fins d'agilité métier maximale.

Mots-clés
Systèmes d'Information Entreprise, Applications, Agilité, Logiciels Sociaux, Ingénierie Dirigée par les Modèles, Composition de Logiciel.

ABSTRACT

Present enterprise information systems are centered on heavy corporate applications, which cannot and indeed do not provide the agility required to survive in todays' competitive business landscape. Actors (business units, individuals, teams and communities) must introduce their own applications to work around these limitations, resulting in a fragmented, inconsistent and ungovernable information system.

This thesis proposes an alternative enterprise architecture paradigm based upon a finer-grained decomposition of information systems and a different distribution of responsibilities. It empowers all actors to contribute fragments to the information system, private or shared with other actors, which can then be composed to form profile-specific applications. Consumer-space social mechanisms are proposed to manage the potentially huge resulting numbers of fragments emerging from the employee community.

The aim of social information systems is both to improve the overall consistency and governability of the enterprise information system and to leverage the collective intelligence and energy of the corporation towards maximum business agility.

Keywords
Enterprise Information Systems, Business Applications, Agility, Social Software,
Model-Driven Engineering, End-User Development, Software Composition.