

Towards Social Information Systems

Marc Quast
University of Grenoble
France
marc.quast@imag.fr

Jean-Marie Favre
OneTree Technologies
Luxembourg
jean-marie.favre@megaplanet.org

Abstract: The development and usage of complex information systems leads to both technical and human challenges as large numbers of stakeholders with conflicting requirements are involved. Though global consistency must be preserved at the corporate level, software applications have to be adapted to particular needs. However, while individual users have the best knowledge about how to perform their job, they have little influence on the corporate information system. Our experience in industry shows that the “Knowledge/Influence Mismatch” leads to the “Information System Fragmentation” problem: departments, teams or individuals unsatisfied by corporate information systems tend to develop “parallel” ad-hoc applications. To cope with this problem this paper proposes core concepts for Social Information Systems in which much more power will be given to user communities, allowing them to extend existing applications for their particular needs, but also to share these extensions with colleagues; and this in a social way. This approach, based on the notion of “perspectives” linked to people, opens the possibility of a social adaptation and democratic evolution of the information system.

1. Introduction

After the “invention” of writing, which marks the move from prehistory to history and is the first information technology, the “invention” of software is perhaps one of the greatest achievements in the history of mankind [1]. Though we still lack a full understanding of what software actually is, let us consider a few facts.

#1: Software is BY and FOR people.

Software is clearly a creation of people for the benefit of people. To be more precise (1) software is *developed BY* people (a.k.a. *developers*) and (2) software is *FOR* the *usage* of people (a.k.a. *users*). As we will see both the development and usage of software can be considered from a social perspective.

#2: Software runs ON computers.

This comes at no surprise, but as suggested by the very terms “Computer Science“ and “Computer Languages“ this fact has been hiding fact #1 for long.

This paper is based on the experience gained during twenty years of development, integration, deployment and maintenance of *Information System Applications* (as defined below) for various multinational corporations, mostly in the high-tech industry.

- An **Information System** (IS) is a set of software applications running in a particular organization (a few applications to hundreds of them). We define IS as software. A broader definition would encompass people, documents, etc.
- An **Organization** is a corporation, an administration, or big department of either. The size of the organization may vary from a dozen of people to many thousands.
- An **Application** is any data-centric software system enforcing some process of the organization or helping it being more productive, or both.
- A **Stakeholder** is a person whose job is somehow affected by the IS. Beyond users this includes administrators, managers, developers, partners, etc.

This paper focuses on one particular challenge: while an IS should ideally capture all information and processes of an organization in a consistent way, the high number of stakeholders often produces conflicting requirements [2]. Although Applications typically provide adaptation mechanisms, we think these are limited and do not provide adequate support for serious conflicts, nor for business agility [3, 6]. We believe new technologies are needed to empower users with social adaptation of Applications and propose enabling concepts for what we call *Social Information Systems*.

A running example will be used throughout the paper. Its simplicity does not reflect the size and complexity of the problem we address, but it exhibits some of its main characteristics. It assumes an on-premise deployment model, but we consider that our proposal also applies to more recent multi-tenant Software-As-A-Service [13] models.

The Example Use Case
<ul style="list-style-type: none">• Company ProjectSoft, a medium-sized software vendor, develops a product called ProjectTracker which provides enterprise project management. This product is installed in more than 100 companies world-wide.• ACME Corporation is a large company which develops both electronics boards and companion software for industrial installations. ACME, a customer of ProjectSoft, uses ProjectTracker for all engineering projects, whether hardware assembly, software development or other activities.• The hardware and software departments have different requirements. However, ACME chooses to share one installation of ProjectTracker (reasons are license costs, people working for both departments, common projects, consolidated dashboards, ease-of-administration, etc...).

The remainder of the paper is structured as following. Section 2 presents a conceptual view of a common adaptation approach based on decision layers. The limitations of this approach are described in Section 3 and Section 4 sketches the principles of an alternative approach more respectful of the social dimension. Section 5 presents the notion of perspective coupled with actors as a basis for implementing Social Information Systems. Related work is described in Section 6, further work in Section 7 and Section 8 concludes the paper.

2. Decision Layers in the Application lifecycle

This section, based on the observation of industry practices, provides a conceptual view of the application lifecycle (see Figure 1 for an overview). Our objective here is not to describe a particular technology but to illustrate the layers in the decision process and to define the vocabulary for the following sections.

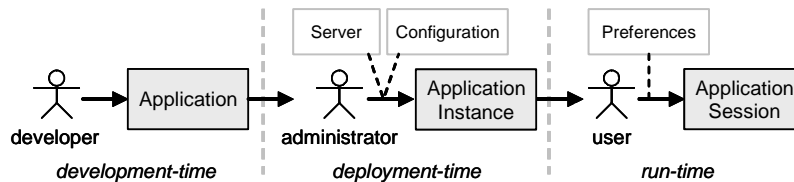


Figure 1. High-level overview of the Application lifecycle

2.1 Development-time – development of Applications

At development-time, an Application can be considered a collection of coarse-grained **ApplicationElements** like **Concepts** (representing entities of the problem domain and structural information, ensuring basic consistency), and **Features** (forms, reports...). Some ApplicationElements 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.

Applications usually define **Permissions** restricting access to ApplicationElements, thus defining logical application subsets. The resulting application reflects the decisions of a (typically fairly small [2]) *software development group* (in the broad sense, including domain experts, ...).

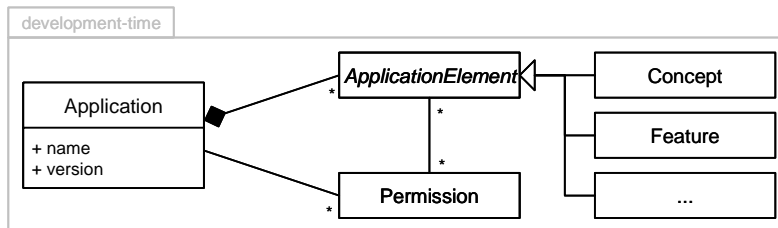


Figure 2. Applications at development-time

Example

- "ProjectTracker 1.2" is an *Application*.
- It provides *Concepts* like "Project", "Task", and "Resource". These can be extended with an ad-hoc mechanism ProjectSoft calls "custom-attributes"
- It provides various *Features* like "TaskScheduler", "GanttDiagramBrowser", etc.
- Its Permissions are {create,read,update,delete}-{Project,Task,...}

2.2 Deployment-time – configuration of ApplicationInstances

At deployment-time, the goal is to solve a specific problem for a given organization. An **ApplicationInstance** is created by physically installing an Application on a **Server** infrastructure and by using ApplicationElement configurability to adapt to the specific purpose. Actual **Actors** (i.e. Users or Groups) are granted Permissions. ApplicationInstances reflect the decisions of yet another small group of *software integrators* and *administrators* - constrained by the earlier decisions of developers.

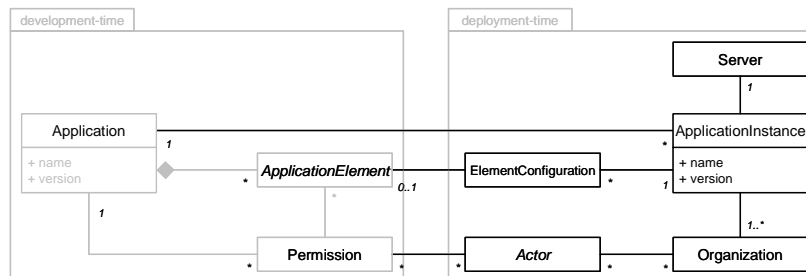


Figure 3. ApplicationInstance at deployment-time

Example

- The “ACME Engineering Project Referential” installed on “http://project.eng.acme.com/” is an *ApplicationInstance* (using “ProjectTracker 1.2”)
- Several Concepts have been extended using the custom-attribute mechanism
 - A custom-attribute “customer” has been added to Concept “Project”
 - For software activities, a custom-attribute “type” with possible values {SPECIFICATION, DEVELOPMENT, TEST, OTHER} has been added to Concept “Task”
 - For hardware activities, it is required to track machine-time consumption, leading to several other custom-attributes and a dedicated usage report
- Several roles have been set up, for example Role “technical-staff” has Permission to see all “Project” and “Task” attributes except “*Cost” and cannot see “Resources”

2.3 Run-time – execution of end-user Sessions

At run-time, a User connects to an ApplicationInstance establishing a **Session**. A Session filters available ApplicationElements according to Permissions, and can provide a final, typically shallow layer of **Preferences**.

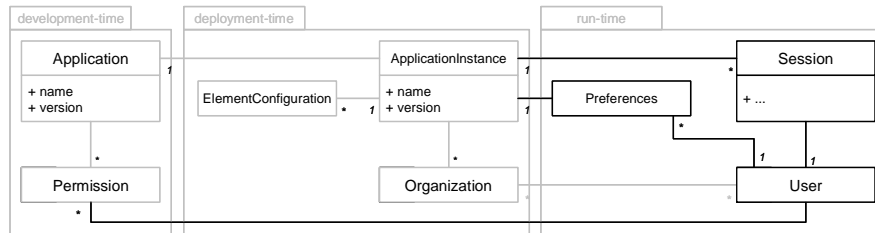


Figure 4. Sessions at run-time

Example

- Fred is a software architect, with role “technical-staff”. His Preferences hide all “Task” attributes except “name” and “duration”
- Betty is the director of sales. Her Preferences show only “name” and “customer” attributes for “Projects”, plus all DATE attributes

2.4. Summary

Users reach the functionality of an Application through a number (possibly higher than in our example) of successive adaptation layers, the intent of which 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 main benefits are cost-reduction through factorization (re-use), and consistency (the adaptation at each level is constrained by what has been decided at the lower levels). However, the approach presents serious limitations as described in the next section.

3. The Knowledge/Influence Mismatch and IS Fragmentation

The problem we have observed with adaptation layers is that the most important decisions are taken by the people farthest away from the real-life problems the application needs to solve. Though 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, although they usually have insufficient knowledge on the details about application usage in particular contexts. We refer to this phenomenon as the **Knowledge/Influence Mismatch**. Agile methodologies [5, 7] have alleviated this problem by reducing the cycle-time through the process in Figure 1, but have not solved it.

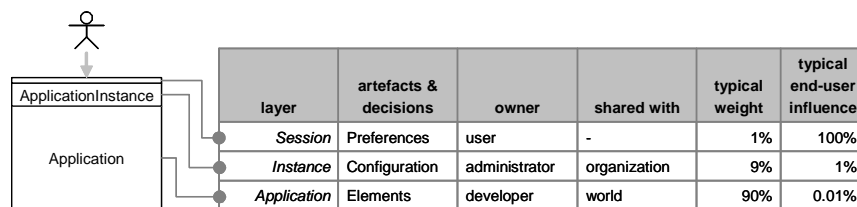


Figure 5. Typical weight of decision layers versus end-user influence¹

The diminishing “end-user influence” figure is not only related to distance to key development roles but also to the fact that the lower layers are shared with a high number of other stakeholders. This sharing is the root cause for the difficulty of *requirements engineering*. Ideally, requirements analysis should be a highly *social* activity [2], bringing together all stakeholders, letting them exchange their requirements

¹ The weight and influence figures roughly represent our experience and are not backed up by empiric data

and expectations and converging on a common solution. Even with present communication technologies, this is an impossible feat, hence the common practice of expecting a subset of representatives to faithfully express requirements of other stakeholders, and as illustrated previously, trusting a few central roles to consolidate this into a single, consistent, satisfactory whole.

The results are that during each iteration, numerous requirements are excluded from both the Application and the Instance layers, that serious compromises on requirements are commonplace and that over time this leads to an overall state of disappointment and resignation of Users with their unsatisfactory information system [6, 7].

We think *Information System Fragmentation* is a direct consequence of this situation. High-end fragmentation happens when dissatisfied, resourceful actors develop their own ad-hoc systems as described in [8], where they have full control over the developer (i.e. influence on the Application) and a reduced number of stakeholders (i.e. fewer or no conflicts). At the low end of the spectrum, spreadsheets are commonly used to store critical data outside of mainstream applications.

Example

- | |
|--|
| <ul style="list-style-type: none">• Fred likes to complete tasks with highest risk first. He keeps a list of his tasks in a spreadsheet, with columns "Task.title", "Task.risk", "Task.notes"• Betty needs a history of project delays to help her sales staff negotiate with customers. She has taken a summer intern to develop a small intranet system on sales.acme.com which extracts project data from "project.eng.acme.com" (via ProjectTracker web-services), adds some sales-specific data and computes nice indicators. This application has quickly become vital for the sales team |
|--|

We believe that multiple adaptation layers are possible without sacrificing influence nor generating conflicts, by introducing a social dimension as described in the next section.

4. An alternative distribution of responsibilities

Actors should be fully empowered to adapt an application to their specific problem, instead of having to rely on external factors like the availability and willingness of central roles and having to reach agreements and compromises with other Actors. As a related example of such a change in the social dynamics of groups, wikis have replaced the formerly powerful, central "webmaster" role with an infrastructure where anybody can freely contribute.

Beyond mere technical platforms, wikis represent a real shift of responsibilities. They imply *trusting the user community* to do what is right for the overall system. This trust should in theory be natural in an IS context because all users are authenticated (typically through an enterprise directory), and most of them are experienced professionals.

However, we think it is unrealistic to expect an organization (community) to agree and converge on all application elements. Divergence of points-of-view is unavoidable in complex information systems [2, 9], and we think that unlike both existing applications and wikis, actors should have a private space, following the sandbox principle.

This implies isolating Actors from each other by default, as opposed to implicit sharing. Sharing with other Actors, both in- and out-bound, should happen explicitly when desired. We can thus summarize the high-level requirements for a new approach as follows.

Actor Bill-Of-Rights		
	Any Actor can as opposed to having to ...
1	choose a subset of available ApplicationElements	daily wade through elements not relevant to his problem
2	extend / add ApplicationElements	appeal to central authorities, comply with more influent peers and/or build a parallel system
3	share whatever he owns with other selected Actors	extract values and manipulate them outside the application
4	override any value	

Independently of aforementioned usability, some end-users might not have the skills to exercise the rights above. However, we believe granting everyone the permission is far more interesting than the present opposite situation.

5. Towards Social Information Systems

We define **Social Information Systems** as ISs built with the *explicit* social dimension in mind with *dedicated* software technology. In this section we present one of the core concepts of our proposal, namely the introduction of the concept of Perspective as a central building block of Social Information Systems.

A **Perspective** defines a specific point of view for one Actor. It defines some ApplicationElements (for either private use or shared with other Perspectives), imports other relevant ApplicationElements and possibly extends them. In our proposal, the concepts of Application and ApplicationInstance are unified into a collection of ApplicationElements specific to a Perspective. Extensions provide a generalization of Configurations.

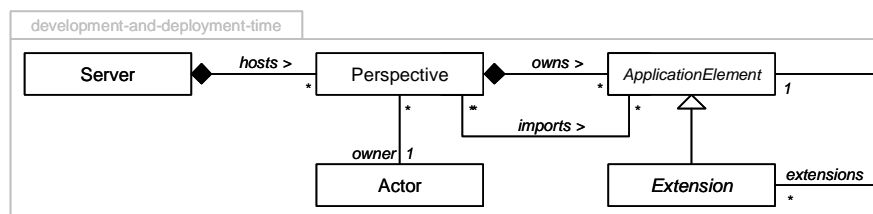


Figure 6. Perspective-centric application meta-model

Example

- "http://project.eng.acme.com/" now hosts an "EngineeringProject" *Perspective*, importing several (but not all) elements from "ProjectTracker 1.2"
- "SoftwareEngineering" is now a specific *Perspective*, which extends Concept "Task" as previously, clearly distinct from the "HardwareEngineering" concerns

When considering social aspects, the notion of **Group** naturally appears. We therefore introduce it explicitly in our model as a refinement of the concept of Actor. In a social context Users typically belong to several Groups (Figure 7). As Actor is the generic term for both concepts, Perspectives are linked not only to Users but also to Groups (Fig. 6).

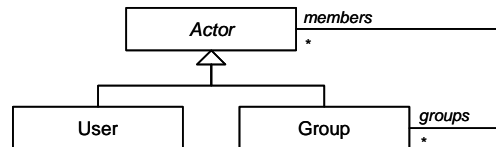


Figure 7. Actors, Groups and Users

It is important to understand that while both Groups and Users are involved at deployment time, at run-time only Users run Sessions. The process of building a session is not trivial, but below is a simplified view that may help in understanding the concept of Perspective. When a User, member of multiple Groups, connects and thus initiates a session, this involves the following steps.

1. fetch the User’s Perspective plus those associated to all of his Groups,
2. fetch all ApplicationElements and related Extensions from these Perspectives,
3. weave (consolidate) these elements into ComposedElements (figure 8).

The whole process results in a Session which constitutes indeed a *virtual private application* built on-the-fly.

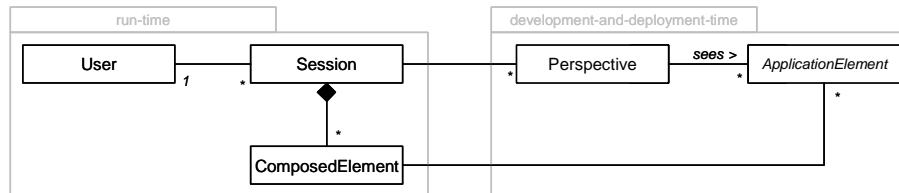


Figure 8. Session and Perspective at run-time

Example

- Fred now defines his own Perspective
 - It extends “Task” with columns “risk” and “notes”, replacing his former spreadsheet
 - Other architects are interested in Fred’s risk-driven approach, and choose to adopt (i.e. import) his “risk” extension
- Betty’s department has its own “sales” Perspective too
 - The summer intern has developed an ApplicationElement for the sales Perspective, which re-uses Project data directly
 - Customer is now a full-blown concept owned by the sales Perspective
 - The other fields and indicators are now mostly extensions of ProjectTracker concepts

Perspectives thus unify development-time artifacts, deployment-time configurations and run-time preferences, providing a successive refinement capability without Knowledge/Influence Mismatch or conflicts.

6. Discussion and related work

Our approach empowers users to *contribute* to the IS and to its evolution. Similarly to web 2.0 systems, users could interact through extending, tagging, rating, and sharing application elements. We believe that this would allow organizations to –continuously– leverage their *collective intelligence* and would enable *social software evolution*, opening new opportunities for social interactions and social schemes. For instance, in the context of open communities, one can imagine an *anarchic software evolution* where everybody can contribute to the same application without any supervision but with the opportunity to share elements through auto-organization. As in all social systems, some more structured organizations typically arise at some point leading for instance to a *meritocracy*. In even more structured contexts and in particular in business organizations, one can imagine *democratic software evolution* supported by the implementation within Social Information Systems of voting mechanisms [2]. We believe that many social schemes that characterize existing social systems could be applied in the software world, especially with appropriate supporting technologies.

The notion of perspective is clearly connected to the notion of viewpoints. However, the study of viewpoints [9] is mostly focused on *reconciling* divergent concerns at the specification and design level. Our proposal takes a complimentary bottom-up approach, allowing and even encouraging divergence in the implementation itself. Private elements allow *all* Actors to implement their specific requirements without complex mappings, lengthy negotiation or forking new ad-hoc Applications. We believe this encourages creativity and exploration, allowing new aspects to *emerge* from the bottom up.

Although our proposal in essence describes technical principles, the strong emphasis on people and groups ties it to the fields of collective intelligence [10] and complex adaptive systems [11].

The work presented here also relates to End-user development, a domain which has been studied intensively [8,12]. We believe that our proposed private elements could help by isolating Actors from one another's experiments. Also, the data-centric nature of ISs should make them less challenging to extend by end-users than more function-centric software; even without full end-user programming capability, we think end-user configuration and modeling would already provide huge benefits.

It is not clear how willing end-users would be to extend applications. We think this will essentially be related to ease-of-use, and are aware that end-user manipulation of the data, the underlying model and the views (forms, reports) represents a major usability challenge.

One can also wonder how *consistent* the resulting IS would be. We could argue that the consistency of most information systems is only superficial. In our experience, corporate-level applications, forming the tip of the iceberg, are usually consistent among themselves. This would not be different in our approach. The remainder however, hidden today in numerous parallel ad-hoc applications, is inconsistent by construction. Worse, it is out-of-reach for analysis. We think an infrastructure with a built-in divergence capability allows for processing to detect convergence opportunities [4,9] and notifying actors of similar-looking extensions, hence our evolutionary development [7] claim.

We also think our proposal fits within a natural trend, since the history of programming languages (PL) has been driven by social considerations [1]. For instance modular languages have been invented to support the development of software by *many people*, some of them using module interfaces, others providing their implementation. Similarly, the keywords “public” and “private” of object-oriented (OO) programming languages (and in the case of C++ the term “friend”) belong to terminology used in social contexts. In OOP these terms refer to the relationships between ApplicationElements, and are mostly used at development-time. Perspectives go one step further by explicitly relating ApplicationElements to people (Actors, figure 6), and by utilizing these relationships at run-time as well.

7. Prototyping and Future work

Building upon the concepts presented in this paper, we are implementing a prototype of a distributed data-centric social infrastructure with the following high-level components.

- a *directory* with users and groups, annotated with references (URLs) to their associated Perspectives,
- a Perspective-centric *repository*, hosting class definitions (types) and their extensions as well as their associated instances (values),
- a *browser* communicating with both to compose simple virtual private applications allowing data and model manipulation.

Communication is asynchronous, standards-based, and resource-centric (REST architectural style [14]). The goal of this prototype is to validate the technical feasibility of our proposal, and to conduct field experiments in both academic and industrial organizations.

From a more theoretical point of view, we are extending the object-oriented paradigm with social concepts through the notion of Perspective. We think *Social Objects* are the building blocks of Social Information Systems.

8. Conclusion

In this paper we have introduced the notion of Social Information Systems. We have showed that the Knowledge/Influence Mismatch leads to users' frustration and thus to IS fragmentation. Our proposed solution is to empower users to contribute to the IS enabling *social software evolution*. We have provided a high-level description of an isolation mechanism called Perspectives to make this possible. Although more work is needed to assess the feasibility of our proposal, we are confident it is a track worth exploring and think that sharing these early ideas is a form of social software engineering in itself.

9. References

1. J.M. Favre, D. Gasevic, R. Lämmel, A. Winter, "Introduction to the Special Issue on Software Language Engineering", *IEEE Transactions On Software Engineering*, November/December 2009.
2. S. Lohmann, S. Dietzold, P. Heim, N. Heino, "A Web Platform for Social Requirements Engineering", *Software Engineering 2009 – Workshopband*, Köllen, Germany, 2009.
3. A. Wensley, E. van Stijn, "Enterprise Information Systems and the Preservation Of Agility", *Agile Information Systems*, 178-187, Elsevier, 2007.
4. N. Ahmadi et al, "A Survey of Social Software Engineering", *23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*, 1–12, 2008.
5. K. Beck et al, "The Agile Manifesto", <http://agilemanifesto.org/> - visited January 2010.
6. S. Newell, E. Wagner G. David, "Clumsy Information Systems : A Critical Review of Enterprise Systems", *Agile Information Systems*, Elsevier, 2007.
7. P. J. Denning, C. Gunderson, R. Hayes-Roth, "Evolutionary System Development", *Communications of the ACM*, vol. 51, 12, 29-31, December 2008.
8. W. Harrison, "The Dangers of End-User Programming", *IEEE Software*, vol. 21, 04, 5-7, July/August 2004.
9. M. Sabetzadeh, A. Finkelstein, M. Goedicke, "ViewPoints", *Encyclopedia of Software Engineering*, P. Laplante, Ed. New York: Taylor and Francis, 2010.
10. J. Surowiecki, "The Wisdom of Crowds", ISBN 978-0385503860, 2004.
11. http://en.wikipedia.org/wiki/Complex_adaptive_system - visited February 2010.
12. A. Sutcliffe, "Evaluating the costs and benefits of end-user development", *ACM SIGSOFT Software Engineering Notes*, vol 30, July 2005.
13. K. Bennett , P. Layzell , D. Budgen , P. Brereton , L. Macaulay , M. Munro, "Service-based software: the future for flexible software", *Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, December 2000.
14. Roy T. Fielding et Richard N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology*, no. 2, 115-150, 2002.