A Social Software-based Coordination Platform
Tool Paper

Davide Rossi

Computer Science Department - University of Bologna - Italy
rossi@cs.unibo.it

Abstract. Organizational best practices are unstructured, emergent processes that freely coordinate actors engaged in reaching organizations' goals. In recent years we are witnessing the wide adoption of social software (blogs, microblogs, wiki, forums, shared calendars, etc.) as primary technological tools to support organizational best practices, fostering their creation, evolution and sharing, allowing their continuous refinement and alignment with the organization's mission and evolving know-how.
While organizational best practices and social software tools are good candidates to support specific processes within the organization (and among organizations) they also present several issues, when compared to classic BPM tools - those based on structured coordination and well-defined process models: since they have no explicit representation it is hard to analyze them (by analytic techniques or by simulation), to monitor their evolution and to support their execution; moreover it is hard to extract explicit knowledge from them.
In this paper we present a set of tools that complement social software in creating a real coordination platform, mitigating some of the aforementioned issues.

1 Introduction

Coordination can be structured or emergent. By this we mean that coordination can be based on the idea of enforcing/supporting interaction patterns among actors on the basis of a well-defined model or can be the result of independent agents defining (and refining) their interactions in an emergent way.

BPMSs (Business Process Management Systems) and social software are instances of these two models: BPMSs support process monitoring and enactment on the basis of a process model defined by some kind of modeling language or notation whereas social software is an emergent coordination *facilitator*. Social software supports social interaction and social production and raises the level and scope of the interaction facilitated by computer and computer networks [9]. It uses a self-organization and bottom-up approach where interaction is coordinated by the collective intelligence of the individuals; the latter do not necessarily know each other and are not organized a priori in a structured way. By publishing and processing information in blogs, microblogs, wiki, forums; by using tagging

services; by collaboratively editing documents, users reach organizational goals following sequences of activities that have been refined in previous interactions.

A BPMS is a coordination platform; social software, per se, is not. Social software provides a set of basic tools to enable information sharing and exchange but provides no support for automating interaction patterns. In this paper we present a set of tools that, combined with social software, implement a coordination platform. We aim, specifically, at a platform supporting *organizational best practices*: the unstructured, emergent processes that freely coordinate actors engaged in reaching organization's goals by interacting with social software tools.

This paper is structured as follows: in the next section we describe how social software can be augmented to become a coordination platform; in section 3 we describe the tools we designed. Section 4 contains a case study that shows the platform in action. Section 5 introduces the coordination model that underpins the platform. Section 6 discusses our approach. Section 7 concludes the paper by presenting some related work and possible future enhancements.

2 The Platform

Coordination allows actors (persons and software systems) to share information and synchronize their activities. At a very basic level of analysis we can argue that social software is not a coordination platform in which, while offering a way to share information, it lacks the ability to synchronize actors. When social software tools are used in a process in which, for example, user A has to wait for user B to complete a given task before resuming their activities, it is responsibility of user A to realize that user B completed their task (which is typically performed by checking the information shared using the social software tools). In other words, users have to manually extract the relevant state information in order make their processes progress. Moreover social software does not provide any method to automatize sequences of activities, even when they are basic parametric sequences of interaction via a web browser; while automatization is not a basic requirement for a coordination platform in itself, it is evident that this ability is essential in order to provide support for organizational best practices.

These observation lead us to the design of two tools, InFeed and WikiRecPlay, whose role is, respectively, to provide mechanisms to extract/manipulate information from web applications and to record/replay parametric sequences of interactions with web applications. The interplay between these tools allows users to define sequences of parametric activities (performed on social software) that can be synchronized with other actors' activities (monitored by extracting relevant state information from social software).

Our goal is to support organizational best practices and it would have been unreasonable to build a prescriptive coordination system, that is a system that enforces coordination patterns; moreover we wanted to support the sharing of best practices and this can be facilitated by sharing information extractions/manipulations and parametric interaction sequences. To achieve this the

two tools are themselves integrated with a social software tool (a wiki that is used as a repository extractions/manipulations and parametric interaction sequences) and can provide recommendations to the users on the basis of the currently visited page and the information stored on the wiki. The user can then decide to adopt a recommendation, add it to their favorites and, eventually, make its firing automatic.

3 The Tools

3.1 WikiRecPlay

WikiRecPlay is a Firefox extension that allow users to record and re-play sequences of web activites (interactions with web sites using a browser). The way users perform such activities has been subject to changes in the recent past: web applications are getting more interactive, ubiquitous and easy to use; the social dimension has become crucial: different users —with different skills and tools— share content easily and complete tasks together, in a new and spontaneous way. From a technical perspective, monolithic server-side applications are being replaced by Ajax-based ones that load and manipulate (pieces of) content client-side. WikiRecPlay has been designed to support users in automating web interactions within this context.

In order to define what we wanted from WikiRecPlay we selected a number of test cases built around known web applications [1]: *GoogleDocs* for its very sophisticated interface and Ajax-based machinery; *MediaWiki* and *WordPress* for their relevance as social software tools; *PizzaBo* and *JQueryUI* for the large amount of highly dynamic client-side code.

WikiRecPlay has been built on an *event-based* model, in order to work on highly dynamic web pages: the application is able to record and re-play the events occurring in the browser (mouse click, form filling, selections, etc.). An alternative approach would have been to capture, store and reply HTTP transactions but such an approach cannot cope with (client-side) dynamic pages, leaving all Ajax-based applications unsupported.

Figure 1 shows the main interface of WikiRecPlay. The sidebar lists all loaded sequences and allows users to edit or replay each of them. A new sequence can be recorded and stored through the same interface.

Figure 2 shows the interface for editing a sequence. Once it has been recorded, in fact, its details appear in the 'Step list' panel. Each step can be configured, moved or deleted separately.

Each step is associated to an event occurring on a page element. The interface shows a screenshot of the page highlighting that element with a red bordered rectangle, and allows users to decide:

– which event needs to be captured

[1] http://docs.google.com http://www.mediawiki.org http://wordpress.com http://www.pizzabo.it http://jqueryui.com

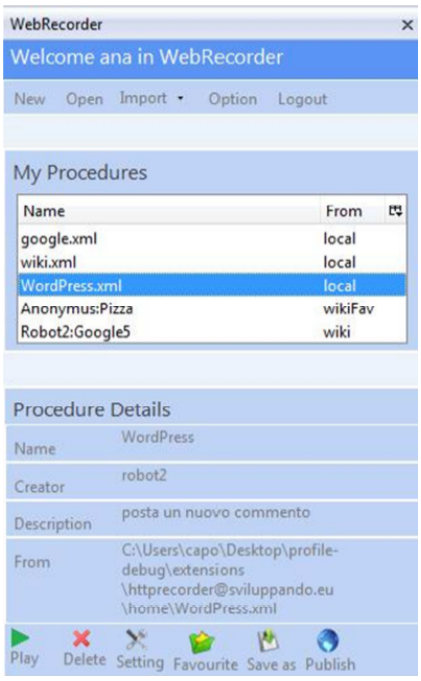

Fig. 1. The sidebar

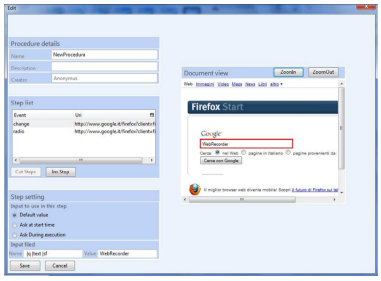

Fig. 2. Configure sequence steps

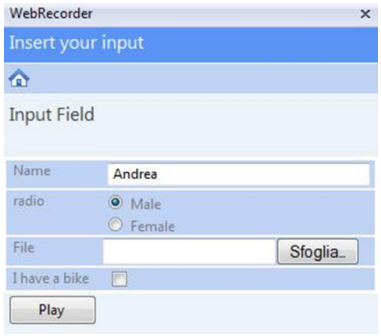

Fig. 3. Set parameters before replaying a sequence

- which information users are expected to provide
- how event-related data (like the content of a text field) will be set when re-playing the same sequence. Three options are available: (i) *default value*, to use the values originally recorded, (ii) *ask at start time*, to make users provide that information before playing the whole sequence, or (iii) *ask during execution* to let the application stop the sequence re-play and ask the user to provide the required data right before they are used.

All these data are automatically collected by WikiRecPlay when recording a sequence; users can update and customize them at any time.

Figure 3 shows a sample interface for inserting data when re-playing a sequence. Such a dialog is dynamically built by WikiRecPlay from the description of [each step of] a registration.

A relevant feature of WikiRecPlay are *synchronization steps*, these are steps that can be suspended until a given event occurs (or a timeout expires). The event does not have to be necessarily triggered on the same page and can be associated to different web applications, like the publication of some content on Twitter, the tagging of a photo on Facebook and so on. This mechanism makes it possible to replay sequences that need to synchronize with activities carried out by different users. In order to support this feature WikiRecPlay can halt a sequence until an XPath predicate on the content of a RSS stream changes from false to true (see Sect. 5 for a discussion on this topic). InFeed, the second tool presented below, has the ability of producing RSS streams as the result of extractions and manipulations of data coming from different sources (other feeds, web applications like social software tools or services like microblogs and e-mails) and is thus the ideal companion to WikiRecPlay. Synchronization steps can be inserted in a sequence after it has been recorded and before storing it. When a synchronization step is inserted as the first step of a sequence we call that a *guarded sequence*. Guarded sequences can be inserted (as any other sequence) among the favorites of a user and the user has the option to mark the sequence so that any time its starting guard is satisfied the sequence is automatically replayed (or *fired*).

WikiRecPlay also allows users to share sequences. Users can store data in two places: in a local XML file or on a wiki (which the sidebar is configured to communicate to). Wikis make it possible not only to easily share sequences but also to edit and improve them collaboratively.

Another relevant feature of WikiRecPlay is the ability, given the current web page the user is visiting and the set of sequences stored in the Wiki to propose to the user the execution of all the sequences that start from the current page (possibly filtering only guarded sequences that would be activated). This becomes a kind of recommendation system that improves the awareness of the user with respect to existing sequences that are (possibly) description of (part of) organizational best practices.

WikiRecPlay: Implementation Details WikiRecPlay is built on the standard Firefox extension mechanisms and, in particular, the XPCOM framework.

The overall application follows the *MVC (Model-View-Controller)* design pattern. An internal XML format —whose details are not relevant here— has been defined to describe sequences and steps and is used throughout the application. The main modules of WikiRecPlay are listed below:

Recorder: captures all the activities (DOM events) performed by the user. **Player**: reads a sequence descriptor and replays it, in case asking input data to the user; the player exploits browser facilities to send HTTP requests and to parse responses. **LocalStorageManager**: saves sequence descriptors in a local repository, by epxloiting the browser storage space. **WikiStorageManager**: saves sequence descriptors on a wiki. This module is in charge of login to the wiki, posting data, retrieving sequences or updating them. It uses the WikiGateway API [14] —that defines a set of operations exported by multiple wiki clones— so that WikiRecPlay is not bounded to a specific server-side platform. **Validator**: validates sequence descriptors, before saving and exporting them. This module actually communicates with a web-service exporting validation features.

While a detailed description of the inner workings of WikiRecPlay is out the scope of this paper we want to highlight that one of the main problems we had to face is related to dynamic web pages: since most elements can be created/moved/deleted at any time it can be tricky (if at all possible) to associate events and current page elements; in several occasions we had to rely on *smart heuristics* to overcome these kind of problems.

3.2 InFeed

InFeed is a feed aggregator/manipulator with an integrated e-mail gateway. It is implemented as a mixed client and server side mashup making use of Dapper[2] (a web content extractor) and Pipes[3] (a visual, interactive feed aggregator and manipulator), both from Yahoo!. With InFeed it is possible to extract data from web applications (this includes usual social software tools but also services like Google Docs, Google Calendar, Twitter, etc...), process them and render them as a feed. The resulting feed can be very terse and easy to parse. For example it is possible to set up a InFeed process that generates a simple "run, I'm away" item in a feed when a Google calendar alarm e-mail has been received and the user tweeted "#infeed away" (after any eventual previous "#infeed available" tweet). This simple feed can easily be used in a synchronization step in WikiRecPlay and let a sequence being played automatically.

4 A Case Study

Consider the following organizational best practice. A group of bird watchers (that interact by participating to a public forum) decides to set up a photographic context. In order to run the context the forum itself will be used: a

[2] http://open.dapper.net/
[3] http://pipes.yahoo.com/

new section is created (e.g. "photo contests"); each time a new contest is run, a thread is created in this section (e.g. "photo contest for the month of May"). The user who created this thread is the contest manager. The contest manager, in the first post, details the subject of the contest (e.g. "eagles in the wild"). Participants have to submit their photos by replying on this same thread; their post have to include a link to the image and an embedded Goggle map detailing the place where the photo was taken. Once the submission period is over the manager locks the thread and starts a poll. The poll runs for a period of time after which it is closed and the manager announces the winner by editing the first post of the contest thread.

This is a glaring example of emergent coordination: users defined how to interact with social software tools in order to complete the *photographic contest process*; no formal description of the process exists but all participant are expected to follow a best practice. In case of anomalies (e.g. too few votes received) it is easy to modify the process (e.g. ask the participant to vote for others' submission). Notice that while we are giving a rather detailed description of the workflow, still this is not a well-defined process in which no formal description of it exists, since this is the result of emergent behavior, and it is very well possible that it will be freely subject to refinements and modifications in future iterations of the contest.

Our platform can support users in participating to this organizational best practice: sequences can be recorded and shared with respect to the various actions required: open a new contest, submit a photo, vote, and so on. These sequences can be used to automate some of the more time consuming (and boring) actions, like submitting a photo, by allowing users to replay (in a parametric way) the sequence in which the user first has to submit his photo to a photo hosting site (like Photobucket), retrieve the URL to access it from outside, connect to Google maps, enter the coordinate for the place, retrieve the HTML fragment to embed the map then, at last, connect to the forum, identify the active contest thread and post the submission. WikiRecPlay can also assist new users in which it has the ability, once users enter the contest thread, to suggest them that a "submit photo" sequence is available, thus allowing them to participate to the contest even if they are not aware about the rules that the community decided. Other useful sequences include, for example, close a contest thread and create a poll. By adding a synchronization step at the beginning of the sequence and setting up a InFeed process as explained in section 3 it is possible to let this sequence fire automatically when a Google calendar signals an event (so the manager just has to set up the correct event in its calendar and can forget about closing the contest manually). It is even possible for a user willing to participate to the next context, whose subject has been anticipated, but who is going to be away with limited connectivity in the period when the context is be run, to prepare his submission and let the corresponding sequence fire automatically when he tweets "#photocontest submit".

Users, by creating and sharing these sequences (that are generally created for their own benefit, to automatize repetitive/boring interactions) concur to

the spreading of organizational know-how. Several experiments [8] have been conducted on using groupware tools within organizations in order to share how-to knowledge but most failed because users have no immediate gain in publishing their knowledge (to the contrary, they feel they are wasting time); with this respect our platform elicits user participation by giving them immediate benefits.

5 The Coordination Model

Up to this point our description of the platform focused on its usage; this decision postponed a discussion to its underpinning model for the last part of the paper. While unusual, we believe this decision helps in better assessing its relevance in the context for which the platform has been designed. In this section we present a more formalized view of the adopted coordination model.

First of all we introduce the concept of process state that we previously informally hinted. Please notice that in this section we assume for process the broad definition of a coordinated set of activities leading to a goal (and not, for example, the instance of a process model), a definition that includes organizational best practices. In our context the state of a process is the combination of all the data related to the process, data that can be scattered through the various social software tools (like blog posts, twitter messages, RSS feed items and so on) and emails exchanged by the actors involved in the process.

Actors pursue their goals through sequences of interactions with various web applications; these sequences are composed of steps; each interaction step results in a modification of the process state (of course there are interactions between the actors and the tools that does not result in a state modification, we simply do not take these into account here). We can then represent a sequence through its steps:

$$a_1, a_2, ..., a_n$$

Some of the steps can be freely performed after their preceding ones has been executed; others require that different actions in the process are performed before being activated. A typical example of this behavior is that of a scientific journal editor waiting for three reviews from different reviewers to be received before deciding whether to accept or reject a submitted article (using social software tools we can support this process using a forum and an organizational best practice that suggest that reviews should to be posted as replies in a thread where the submitted article is attached to the first message). We make these synchronization requirements explicit in the sequence by introducing synchronization steps. These steps halt the execution of a sequence until the process reaches a specific state (or, more precisely, until a condition upon a subset of the state is satisfied). In the aforementioned example the synchronization steps that halts the sequence waiting for the three reviews to be posted is satisfied when the number of the posts in the submission thread (that is displayed in the forum web interface) reaches the value 4.

By denoting with s a synchronization step, the sequence becomes:

$$a_1, ..., a_j, s_1, a_{j+1}, ..., a_k, s_2, a_{k+1}, ...$$

We can add a dummy sequence step at the beginning of the sequence and split it at the synchronization steps obtaining sub-sequences of the form:

$$s_1, a'_1, ..., a'_{n'}$$

$$s_2, a''_1, ..., a''_{n''}$$

...

By adding a causal requirement to each step $s_1, ..., s_n$ (in order to impose the sequential activation of the sub-sequences) we produce $s'_1, ..., s'_n$ that we use to replace the original synchronization steps in our sub-sequences.

The sub-sequences thus obtained are rules in which the first step is a guard and the following ones are actions that change the state of the system. The use of state-based rules to realize coordination belongs to several well-known coordination models, languages and systems: this is the case of Gamma [3] -inspired languages (such as the CHAM [4]), of Interaction Abstract Machine [12] -inspired languages (such as LO [1]), of blackboard-inspired languages (such as (Extended) Shared Prolog [6]) and, to some extent, to Event-Condition-Action-based workflow execution engines too (such as the one described in [5]).

It should be noted, however, that while most of the aforementioned proposals assume a rewriting approach in which the rules (atomically) consume and produce elements of state, in our approach the guards do not consume state elements but simply check a state-based predicate. One of the main consequences of this approach is that, if no countermeasures are applied, once a rule has its guard satisfied that rule can fire an indefinite amount of times until the predicate associated to the guard becomes false. In order to avoid this behavior, as described in Sect. 3, rules are activated only when a predicate associated to a guard changes from false to true which means that, technically, the rules are based on a state-transition event. Another relevant issue to keep in mind with respect to the coordination model and its actual implementation is that our systems realizes a *coordination overlay* on top of social software and, as such, it inherits most of its limitations. This means that there is not a synchronized view of the shared state and locking is not available (since is not provided by the underlying system), thus it is not possible to guarantee transactionality, atomicity and mutual exclusion. Consider also that state changes are not notified by the Web applications and our system has to recur to polling (which amplifies the state-view synchronization problem).

While these limits are significant the reader has to keep in mind that this system has been designed to support (and sometimes replace) the users in their interactions within Web applications and, as such, these are the very same limits human users have to cope with.

The coordination model we just introduced is quite similar to the one proposed in X-Folders [13]. The differences in the platform, however, are noteworthy: X-Folders operates on information stored in document spaces and actions are sequences of Web service calls.

In general we argue that the use of a rule based coordination model in the context of social software is quite natural: the fact that actions depend on a shared state and not on the state of singular actors and the fact that interaction patterns are not imposed from the environment (coordination is endogenous, not exogenous [2]) clearly point to rule-based models as the better candidates. It is interesting to notice that the structured/emergent dichotomy we cited in Sect. 1 is related to the one between exogenous and endogenous coordination languages: exogenous coordination languages (most business process modeling languages and notations fall under this category) are the ideal partners of structured coordination whereas emergent coordination is naturally better addressed by endogenous languages (the astute reader may argue that structured coordination can be addressed with endogenous coordination languages as well; true, but this case in not relevant in the context of this paper).

It is worth to notice that, whereas internally WikiRecPlay is implemented on the basis of the presented model, the rule-based approach is never directly exposed to the end user who can keep thinking in terms of long interaction sequences that are usually easier to understand since users tend to take a personal perspective of the process that ultimately results in the sequence of actions they are in charge of.

6 Discussion and Related Works

Most of the existing coordination systems proposed to complement social software tools are based on a prescriptive approach and usually require the modification of the tools (that, ultimately, means that usual online services cannot be adopted); this is for example the case for [7]. Some research work has also been carried on the idea of sharing interaction sequences for web applications (part of what WikiRecPlay does), CoScripter [10] (and its evolution ActionShot [11]) being notable examples. Just like WikiRecPlay, CoScripter allow users to share recordings into a Wiki to share them. The main differences between WikiRecPlay and CoScripter are: (i) CoScripter encodes user gestures with an easy-to-read scripting language that mimics natural language whereas WikiRecPlay adopts a much more refined user interface; (ii) CoScripter does not support most dynamic pages in which elements are created/modified after the page is loaded in the web browser whereas WikiRecPlay has been designed to support most of these pages; (iii) recordings personalization in CoScripter is implemented by using a *personal database* in which user-dependent data can be stored whereas WikiRecPlay allows the user to personalize recordings by showing dialogs in which instance data can be provided; (iv) CoScripter has only basic support to halt a sequence replay whereas WikiRecPlay can halt an action sequence and resume its execution when a specific event takes place. This last point is possibly the most glaring difference with respect to our approach: CoScripter, in fact, can only be used to replay the interactions of a single user with a web application but cannot be used in the context of multi-user coordinated processes since it lacks support any explicit synchronization support.

7 Conclusions

Social software is an enabling technology for emergent processes. Social software, however, is not a coordination platform in which it offers no support other than making information available. In this paper we presented a coordination platform built on top of social software, that requires no modifications to the existing tools and that plays nicely with the open collaboration idea that is promoted by social software. The platform is implemented by augmenting social software tools with WikiRecPlay and InFeed providing support for defining, sharing, automating interaction sequences and synchronizing users' activities, that is: providing support to share and enact organizational best practices.

Both WikiRecPlay and InFeed, while actual running software, are to be mainly intended as poofs-of-concept, as such they present several limitations. One of the current limits of WikiRecPlay is that it is only available when the user's browser is in execution. This means that automatic guarded sequences are not fired when the browser is not running. While this is a major limit to the actual use of our platform (we acknowledge this, and in fact we are working on a off-line, server-side version of the sequence player) the existing implementation has to be intended as a proof-of-concept and as such it serves its purpose. InFeed does not suffer from the major limitations present in WikiRecPlay, and it is also a much simpler system, since it delegates most of its functionalities to Dapper and Pipes. This also mean, however, that it inherits all the limits of these systems (that are usually restriction with respect to the intended use rather than technical limitations - for example Dapper cannot be used, by design, to extract content from sites that can be accessed only after authentication).

Future versions will enhance the tools and improve their "on-the field usability" but the basic working mechanism are going to be the same of the current proof-of-concept implementations.

References

1. J.-M. Andreoli and R. Pareschi. Communication as fair distribution of knowledge. In *OOPSLA*, pages 212–229, 1991.
2. F. Arbab. What do you mean, coordination? Technical report, Bulletin of the Dutch Association for Theoretical Computer Science, NVTI, 1998.
3. J.-P. Banâtre and D. Le Métayer. Programming by multiset transformation. *Commun. ACM*, 36(1):98–111, Jan. 1993.
4. G. Berry and G. Boudol. The chemical abstract machine. *Theor. Comput. Sci.*, 96(1):217–248, 1992.
5. C. Bussler and S. Jablonski. Implementing agent coordination for workflow management systems using active database systems. In *Research Issues in Data Engineering, 1994. Active Database Systems. Proceedings Fourth International Workshop on*, pages 53 –59, feb 1994.
6. P. Ciancarini. Coordinating rule-based software processes with esp. *ACM Trans. Softw. Eng. Methodol.*, 2(3):203–227, 1993.

7. F. Dengler, A. Koschmider, A. Oberweis, and H. Zhang. Social software for coordination of collaborative process activities. In *Business Process Management*, pages 396–407, 2010.
8. C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, Jan. 1991.
9. S. Erol, M. Granitzer, S. Happ, S. Jantunen, B. Jennings, P. Johannesson, A. Koschmider, S. Nurcan, D. Rossi, and R. Schmidt. Combining bpm and social software: contradiction or chance? *Journal of Software Maintenance and Evolution: Research and Practice*, 22(6-7):449–476, 2010.
10. G. Leshed, E. M. Haber, T. Matthews, and T. Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1719–1728, New York, NY, USA, 2008. ACM.
11. I. Li, J. Nichols, T. Lau, C. Drews, and A. Cypher. Here's what i did: sharing and reusing web activity with actionshot. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 723–732, New York, NY, USA, 2010. ACM.
12. J. marc Andreoli, P. Ciancarini, and R. Pareschi. Interaction abstract machines. In *Trends in Object-Based Concurrent Computing*, pages 257–280. MIT Press, 1993.
13. D. Rossi. X-folders: documents on the move. *Concurr. Comput. : Pract. Exper.*, 18(4):409–425, 2006.
14. B. Shanks. Wikigateway: a library for interoperability and accelerated wiki development. In *Proceedings of the 2005 international symposium on Wikis*, WikiSym '05, pages 53–66, New York, NY, USA, 2005. ACM.