Enabling Community Participation for Workflows through Extensibility and Sharing

Rania Khalaf, Revathi Subramanian, Thomas Mikalsen, Matthew Duftler, Judah Diament, and Ignacio Silva-Lepe

IBM T.J. Watson Research Center, 19 Skyline Dr., Hawthorne NY 10532, USA
{rkhalaf,revathi,tommi,duftler,djudah,isilval}@us.ibm.com

Abstract. This paper describes how community participation may be enabled and fostered in a hosted BPM system. We envision an open, collaborative system, wherein users across organizational boundaries can work together to develop and share design-time and run-time artifacts; namely extension activities, workflow models and workflow instances. The system described in this paper enables this collaboration and also allows the community to provide feedback on the shared artifacts via tags, comments and ratings.

Keywords: Social Software, BPM, Collaboration, Extensions, SOA.

1 Introduction

Workflow languages and systems are steadily making the transition to a more open and collaborative space as the idea of offering business process management systems as hosted services gains momentum. One step in this evolution is the emergence of Service Oriented Architecture, where services from different providers can be integrated using the Web services stack of XML standards [24]. WS-BPEL is the workflow language in the stack, providing a rich flow-based model for aggregating interactions with several Web based services, and in which the workflow is itself exposed as a set of such services. The next step is the move to simpler Web services created using the REpresentational State Transfer (REST) [7] architectural style. REST services are offered with a uniform interface over HTTP and have facilitated the current movement towards Mashups: new services created quickly and easily by aggregating several existing Web based services and visual components. Mashups come in different flavors: User Interface mashups like putting one's running route on a Google map, data mashups like a new RSS feed that combines existing news feeds using Yahoo! Pipes or the IBM Mashup Center, and more recently service mashups [3]. The Bite workflow language [5] is a result of this transition, created to enable fast and easy authoring of workflows that aggregate interactions with RESTful services, humans via forms, e-mails, collaboration software, and back-end services. Bite is by design ready for community participation capabilities, due to its extensible nature and REST based interaction model. The current state in the evolution has been the realization of the software as a service vision, enabled by the move to Cloud Computing [1]. Hence, we now see commercial offerings of hosted BPM systems with pay-per-use models.

The combination of a hosted, multi-tenant enabled BPM system with a lightweight, Web-friendly language like Bite offers organizations and users the ability to design, execute and monitor workflows directly from the Web browser. In this paper, we show how this combination can effectively enable workflow systems to take advantage of the social software paradigm. We focus on enabling social production [2,18] for two specific aspects in a hosted workflow system: (a) extension activities and (b) the workflows themselves. An 'extension activity' is a new kind of activity that is not present in the definition of the workflow language itself. In particular, we enable independent IT developers to easily create and publish extension activities which can be shared with and used by workflow designers and other developers. The community, consisting of both IT developers and workflow designers, can provide feedback on these extensions by using the tagging, commenting, and rating features of a shared catalog. Workflow designers can then create new workflows by selecting from a default palette of basic Bite activities, as well as from a catalog of extensions provided by the community. In addition to this collaboration between workflow designers and developers, collaboration is also fostered between developers by sharing the source code of their extensions and between workflow designers by sharing flow models and/or instances.

Providing this functionality in an open cross-enterprise environment necessitates solving problems in two complementary but distinct areas: (1) Designing a method for social production of worfklows and extension activities and supporting it in the underlying system and (2) providing mechanisms and contracts to cover general software as a service concerns including malicious code, billing, IP issues resulting from reuse, etc. In this paper, we focus on the first area.

The following scenario illustrates a true community experience around authoring and sharing workflows and extensions. In it, we make use of LotusLive (http://www.lotuslive.com), a software as a service offering from IBM providing cross-enterprise collaboration services.

Carol, a LotusLive developer, writes an extension that zips up files in the LotusLive 'Files' file sharing service that are shared with a user within a specified date range. She contributes this extension to the shared catalog, where it gets high ratings.

Brainshare Inc. is attending a career fair hosted by McGrath University where it hopes to attract the brightest student attendees. Ted, Brainshare's event coordinator, invites the Dean of the University to participate in developing the workflow model for the event. The Dean suggests engaging students via a design contest, with the winning design featured in some of the company brochures. Brainshare's existing recruiting workflow model is modified to include additional activities for the contest and to feed the contest results into the brochure design activity. For example, Ted adds to the workflow Bite extension activities for the LotusLive 'Activities' collaboration service [17]. The Dean also informs Ted that all student resumes are available via the LotusLive Files service, but are typically private. On the day of the career fair, interested students will share their resumes with Ted using the Files service. Ted looks at the catalog of extensions and finds Carol's extension. It is just what he needs. He wires that to an email activity so that he can email himself and his HR department a zip file containing the students' resumes.

2 Related Work

We first address related work for social software around shared artifacts on the Web and then focus specifically on workflow. A large array of sites already exists for sharing artifacts via tagging, ranking and commenting such as for photos(Flickr), videos(YouTube) and goods(Amazon.com). More recently, structured content types such as reusable lists [9] are gaining traction as socially shared artifacts types.

A common type of application sharing, specifically among developers, is sharing source code. While this is not new [20], some popular code sharing sites [10] and tools [14] either already include or are working to include social aspects of development. These aspects include public vs. private repositories, project activity graphs, collaborative development tools, etc. While our work is not designed for collaborative development of source code per se, it is leveraged in the context of developing extension activities for process-based applications, as described in sections 4 and 7.

Another popular, but perhaps more minimalist, mode of sharing applications on the web is to index application code [4] or web service APIs [16] available elsewhere on the web and provide community features, such as forums, How-Tos, blogs, etc., around the index. Our service catalog, described in section 7, indexes applications and artifacts that are under the control of the BPM system.

Enabling social software for Web based services is explored in [15] and [21]. In [15], a Wiki-based approach for describing services uses a UDDI [24] registry complemented by a wiki-based semantic annotation subsystem. A service published to the registry contains keywords from the ontology, enabling the resulting wiki page to contain those keywords as well as semantic links obtained by automatic reasoning from the ontology. In [21], service communities are introduced as the combination of social and business communities with the purpose of exchanging services. A dynamic Service Communities platform [6] enables services of interest to be contributed, grouped, consumed, and managed. Drawing a parallel, our work extends that concept to workflows: as users form 'BPM communities' to share workflow applications, they enrich the BPM community platform by adding and refining processes, extension activities, and tags.

The concepts of social software in relation to workflow are described in [18], where social production [2] enabling community contributions in the context of workflow is encouraged. A wiki-based workflow system is described in [8], based on state machine based flows driven by forms. In [12], social networks are created either via a recommender system or a process model repository. This could be layered on our system to enrich the design process by encouraging the reuse of models and model snippets and extended to handle a repository of extension activities.

Examples of commercial BPM as a service systems include Lombardi Blueprint, Serena Business Mashups, RunMyProcess, and IBM's recently announced BPM BlueWorks. Lombardi Blueprint enables modeling and sharing process models with collaborative editing but not execution, while Serena and RunMyProcess also offer execution. RunMyProcess provides a set of pre-configured connectors that seem similar to extension activities. It is not clear from the available description how one can extend this set with new connectors, but seems possible. Note that while WS-BPEL [24] is extensible, the standard does not (by design) cover how extensions can be created or shared. While IBM's BPM BlueWorks enables a host of collaboration capabilities around BPM, it is focused on a higher level of abstraction and therefore does not include direct deployment, hosted Bite flow execution or dynamic addition of extension activities. Its workflow editor and workflow model sharing modules are based on the system in this paper which itself is part of [13].

Our work enables the creation, immediate sharing, and dynamic deployment of extension activities into a BPM as a service design and runtime environment, as well as community participation between users across organizational boundaries via sharing, contributing, and collaborating around extension activities, workflow models and workflow instances.

3 The BPM as a Service System

A 'BPM as a service system' is one in which workflow design and management capabilities are offered as a hosted service accessible via the Internet [13]. In this section, we provide an overview of our system and how users work with it.

Figure 1 shows the modules and roles of the system. The modules are: (1) an Extension Development module for the development and sharing of extension activities; (2) a Workflow Editor module for editing and sharing Bite workflows, deploying workflows into the Workflow Runtime, and monitoring running instances; (3) a Workflow Runtime module that provides a Bite workflow engine

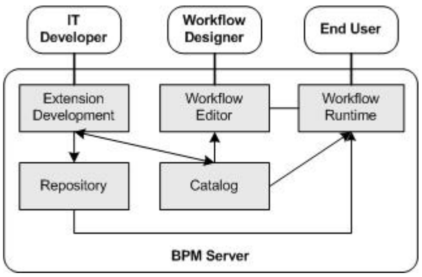


Fig. 1. BPM as a service system and its user roles

with APIs for monitoring and deployment; (4) a Catalog of extension activities; (5) a Repository of extension activity implementations. These modules run in a Web server environment where extension activities and workflow instances are added dynamically to the running system. The roles are: (1) an IT Developer who develops workflow extension activities; (2) a Workflow Designer who designs workflows; (3) an End User who runs and interacts with workflow instances. Interactions with the system occur via REST HTTP calls, so a user requires no installed software beyond a Web browser.

The system can be deployed within an enterprise for internal use, or hosted externally for use by multiple companies. The former allows integration with a company's private user registries, databases, and other internal systems; the latter allows for a larger community of users within which to share workflows and extensions. Additionally, if the system is deployed as part of an external, multi-tenant environment like LotusLive, a user's or company's network of contacts can be used in determining share lists and in searching for other workflows and extensions. This enables cross-company collaboration, such as allowing Ted to use Carol's extension and share the flow with the Dean. The Workflow Editor contains a palette populated with various activities, organized by categories, available for use in composing workflows. The activities consist of both native Bite activities, like 'receiveGET' and 'sendMail', as well as extension activities that have been added to the palette, for example, to perform database operations or consume calendaring services. The set of activities depends on the services the current designer is authorized to consume. Designers develop workflows by dragging activities from the palette to a canvas, wiring them together, and providing each with additional details. For example, the 'GET' activity requires a URL; the 'sendMail' activity requires an email address and subject. The additional details, presented as collections of fields in a properties pane, are populated by typing in values directly, selecting values from pull-down lists, or using expression builders. One property that is common to several types of activities is the list of users/groups/roles that can interact with an entry-point.

Once a workflow model is saved, it can be shared. When a workflow is ready to be run, it is deployed via a menu option. Once deployed, the workflow can be used by end users specified by the workflow designer.

Next, we describe creating and sharing extension activities for such a system.

4 Creating and Sharing Extension Activities

The mechanisms for adding workflow extension activities depend on the flow language and environment in use. In Bite, the developer need only provide an extension activity implementation that will be called by the engine at runtime when the extension activity is reached in the flow. The implementation may be written in either Java or any of a set of scripting languages like JavaScript and python. Workflow extension activities may be developed using various tools, either locally or hosted. The choice of development tool will depend on the complexity of the extension; a simple text editor or Web page form is sufficient for many script-based extensions, while tools like the Eclipse [23] Java Development Tools and AppBuilder [11] will be more appropriate for Java-based extensions. When development tools support features such as collaborative editing (real time or more along the lines of revision control systems), creating an extension activity can become a truly collaborative experience.

IT developers share their work by publishing extension activities to the catalog. For each extension activity, the catalog maintains basic bookkeeping information (e.g., name, owner), implementation details (e.g., schemas and executable code), visibility (e.g., public or private), and informal semantic descriptions (e.g., tags and comments); for complex extension activities (e.g., those implemented in Java) extension implementation modules are stored in the repository. Once in the catalog, extension activities can be discovered by other users. For example, Ted finds Carol's zipUpSharedFiles in the catalog.

In addition to traditional search capabilities, the catalog supports tag-clouds and bookmarking, allowing users to easily discover new extension activities and to organize those that they have previously encountered. Further, bookmarks can include tags and comments, allowing the community to contribute to the description of an extension activity.

The catalog and repository are integrated, via REST-based APIs, directly into the extension activity development tool and workflow editor. This deep integration allows IT developers to publish extension activities directly from their development environment; developers can also import existing extension activities, which can serve as templates or examples for new extension activities. The integration also allows workflow designers to discover available extension activities directly from the workflow editor tool.

In the following section we describe how workflow designers can use extension activities in workflows and deploy the workflows into the environment.

5 Using Extension Activities in Workflows

We will use our scenario to illustrate how a designer uses extension activities. The experience is somewhat similar to selecting an iPhone application from the iTunes AppStore. Once Ted agrees to the Dean's suggestion to pull the students' resumes from LotusLive Files, Ted opens the workflow model and looks in his palette for an activity that can retrieve the resumes from the Files service. Not finding such an activity, he searches the catalog for extension activities that he can pull into his palette to do this work. He finds Carol's highly-ranked extension and clicks to import it into his palette, where it becomes available for use like any of the pre-existing activities with the appropriate property sheets. When the Dean opens the shared workflow with the new extension activity, he has the option to add this extension to his palette as well.

In the case that Ted does not find a suitable extension, he can request one to be created by his company's IT department. Alternatively, he could post a request to an online developer marketplace such as Guru.com or GetACoder.com. One could also extend the catalog site to include such a service geared specifically to workflow extensions.

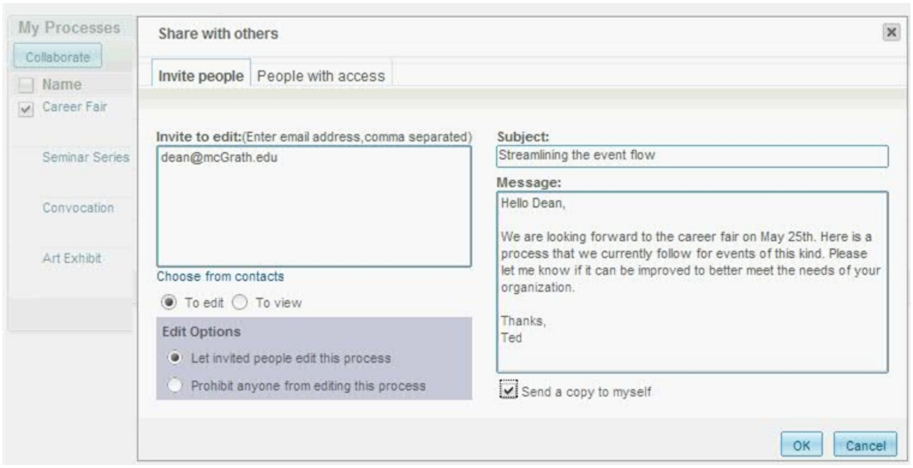
The workflow runtime is integrated into the workflow editor (via REST APIs) allowing designers to deploy workflows directly into the BPM server environment. When a workflow containing any extension activities is deployed, both the extension implementations and workflow are bundled into a Bite workflow application. To do so, meta-data for the extension activities used in the workflow is located using the catalog and used to add appropriate configuration details into the application. For extensions implemented in Java, the implementation modules are retrieved from the repository and, along with any other dependencies, injected into the application. The generated Bite application is then deployed to the BPM server environment, where it can serve requests from end users.

6 Creating and Sharing Workflows

The ability to share workflows is a key enabler for social participation as well as for improving efficiency within an organization or across the cloud. Sharing a workflow consists of sharing the workflow model and/or any of its (running) instances between users within and across enterprise boundaries.

Workflow Models. A workflow model may need to be shared for collaboration, for soliciting comments, for circulating information or for promoting reuse. A workflow model can be shared with all the users in the cloud (public), a group of users (restricted) or nobody (private). Shared models appear in the list of workflows a user may view/edit. There are two types of privileges for each model - Read Privilege dictates who can view or download it and Write Privilege controls who can make changes to it. Figure 2 shows the UI for inviting additional authors via email. In the case of a restricted read or write privilege, a user group needs to be defined. This group can be your company directory, a subset of the company directory, your contact list (which might be cross-company), and so on.

Workflow Instances. Sharing workflow instances is enabled by leveraging the Bite language's addressability model for the workflow definitions, instances, and individual points of interaction (entry points) within a particular instance. This model allows each of the above items to be identified by a unique URI. A workflow definition has a base URI, and each addressable entry-point (i.e. each receive activity) has a path segment that is composed with the base URI to create unique, fully-resolved, URIs. When a particular receive activity is a starter node, an HTTP GET or POST request directed at its associated URI results in the creation of a new workflow instance. The address of this newly-created instance is defined to be the base URI of the workflow definition, with a generated, unique, instance id appended as a URI path segment. This instance-specific URI is returned to the caller in an HTTP 'Location' header (similar to the way the Atom Publishing Protocol works). Any system designed to allow many users to collaborate and share resources must also enable specifying and enforcing security policies. In our system, each entry-point of a workflow can be secured for use only by particular users, groups of users, or users serving particular roles.



Invite to edit:(Enter email address,comma separated)
dean@mcGrath.edu

[✓ To edit] To view

Edit Options
[✓ Let invited people edit this process]
Prohibit anyone from editing this process

Subject:
Streamlining the event flow

Message:
Hello Dean,

We are looking forward to the career fair on May 25th. Here is a process that we currently follow for events of this kind. Please let me know if it can be improved to better meet the needs of your organization.

Thanks,
Ted

[✓] Send a copy to myself

Fig. 2. Inviting users to collaborate on workflow models

The use of a REST-based workflow language makes it very easy to give a community the ability to share and execute the flows. Sharing a flow simply means sharing a link for one or more starter nodes for that flow. Likewise, in order to share a running instance, just provide links to the entry points for the running instance (recall that this URI will include a path segment that uniquely identifies the running instance).

This URI-based mechanism for addressing workflow definitions and instances also makes it straightforward to leverage existing services (e.g. del.icio.us) and/or create a UI specifically for workflows in order to discover and interact with them.

7 Implementation

We have implemented a prototype of the BPM system described above that builds on the BPM as a service platform in [13] and specifically the SOAlive [19] platform and Bite runtime [5] subcomponents to provide an integrated, hosted environment for developing and sharing extension activities, and for modeling, deploying, and sharing workflows that may use such activities.

Figure 3 illustrates the major system components relevant to this paper. All server-side components are built on WebSphere sMash [11]. sMash is an agile Web application development platform, providing a new programming model and runtime that promotes REST-centric application architectures.

Catalog. The catalog maintains extension activity meta-data, including zero or more usage specifications describing how the extension activity is to be used. Such specifications can include human readable documents (e.g., HTML) as well as machine readable schemas (e.g., XML Schema.) The meta-data also specifies the extension activity's implementation. For simple script-based extension

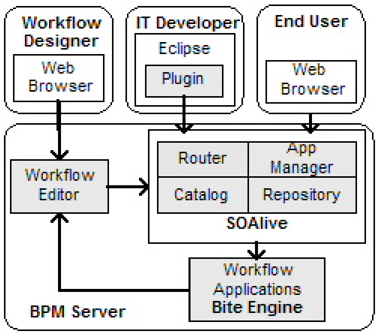


Fig. 3. BPM System Implementation

activities, the implementation script is included in the meta-data. For complex extension activities, the meta-data includes a reference to an implementation module stored in the repository.

The metadata shown below is for Carol's extension activity, which creates a zip archive of all files shared with a given user within a certain date range.

```
{ "name" : "zipUpSharedFiles"
 "display_name" : "Zip Shared Files",
 "description" : "Create a zip of all files ...",
 "uri" : "http://extensions.bite.lotuslive/zipUpSharedFiles",
 "module_id":"lotuslive:lotuslive.bite.extensions:1.0"
 "visibility" : 1, "tags" : [ "Zip", "Files"],
 "schema" : [{"type" :"apiDesc", "value":
     "{'attributes':,{'name':'startDate','display':'start date'}
     ,{'name':'endDate','display':'end date'}]}"} ...] ...
```

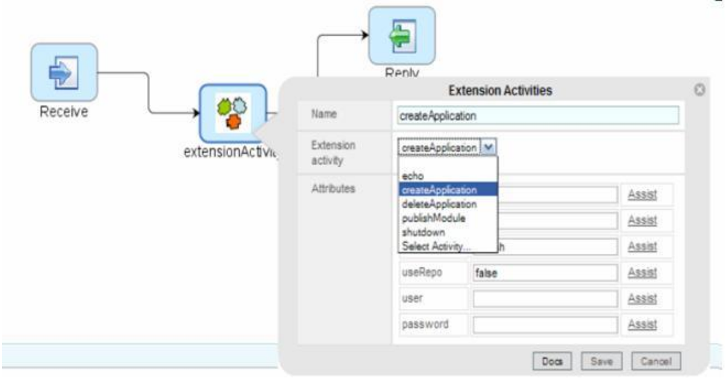
Repository. The repository is an implementation of an Ivy [22] HTTP server providing a REST-based interface for manipulating repository artifacts. Java-based extension activities are packaged as sMash [11] modules and stored as artifacts in the repository. A sMash module can include extension activity implementations (e.g., Java classes,) extension activity meta-data, and dependencies on other modules. Dependencies are described using Ivy meta-data.

Eclipse Plugin. The SOAlive Eclipse plugin extends the sMash Eclipse plugin to integrate intracting with the catalog and repository into the Eclipse IDE. The plugin supports building and publishing extension activities as sMash modules, where a single sMash module may provide the Java implementation of multiple extension activities. The developer declares the names of the extensions and their implementation classes in the file 'config/extensions.config', and creates a JSON meta-data file (in the 'extensions' directory) for each extension. When ready, the developer uses the plugin to publish the sMash module, and its extension meta-data, to SOAlive: the plugin uploads the implementation module to the repository and publishes the extension meta-data to the catalog.

To import, customize, and republish existing extensions, the developer uses the plugin's import wizard to import the sMash implementation from the repository, makes changes, and publishes.

Workflow Editor. The workflow editor allows designers to easily include extension activities in their flows. When an extension activity is used in the workflow, the property sheet of the activity in the palette is modified based on the configurable meta-data of the extension activity (Figure 4). This enables the flow designer to configure the extension and suitably wire it to the tasks that precede/follow it. The meta-data is also used when generating the Bite flow XML.

Currently, one palette item is provided for all extension activities. It gets bound to a chosen extension once it is dragged to the canvas. To support different palette items for each as described in section 5, we are adding an icon image and a category to the meta-data and adding an 'import' option to the palette.



Name: createApplication
Extension activity: createApplication

useRepo	false	Assist
user		Assist
password		Assist

Fig. 4. Workflow with open property sheet of the zipUpSharedFiles extension

Application Manager. The Application Manager is responsible for deploying and managing workflow applications. For a given workflow, the application manager creates a sMash-based Bite application that includes the Bite workflow (XML file), the extensions it uses, and the minimal set of modules needed to execute it. The application manager locates, in the catalog, the meta-data for each extension used in the workflow. Simple script-based extensions are added to the application's 'extensions' directory; Java-based extensions have their implementation modules added as dependencies in the application's ivy.xml file.

Each workflow application executes in a separate JVM process, providing security and fault isolation between running workflows and other system components. Such isolation is critical in hosted, multi-tenant systems, in particular when user-provided code (e.g., a Java extension activity) is allowed to execute. Further, it allows us to easily deploy (and re-deploy) workflow applications into the running system. The sMash runtime model efficiently manages these processes, activating and de-activating JVMs as needed to service user requests. By tailoring each sMash application for a given workflow, we minimize the resources consumed by the workflow when it is active in a JVM.

Bite Workflow Engine. The Bite workflow engine and monitoring module are among the set of modules bundled in the sMash application of a deployed workflow. The engine creates and executes flow instances; the monitoring module forwards flow events (e.g., 'new instance') to the editor. An extension activity is registered with the engine by providing the extension's implementation module and a mapping between it and the activity's name. No meta-data is required. Once an extension activity is reached by the engine's navigator logic, control is handed to the corresponding implemenation module. The module gets access to the activity definition, with any expressions already resolved. Once the module completes its work, it hands control and any output data back to the navigator. The navigator places the data in the activity's output variable, marks the activity complete, and continues. The workflow instance itself is a REST-based resource.

Router. The router enables a cloud computing environment for running workflows, where each workflow application is installed on one of several available nodes in the cloud. Acting as a reverse proxy, the router manages HTTP connections between end users and workflow applications. Each deployed workflow is assigned an external URL prefix that uniquely identifies the workflow application; the router is responsible for maintaining the mapping between these external URLs and the locations of installed workflow applications, and for forwarding requests from end users to workflow applications. Our prototype supports a pluggable cloud architecture where different cloud environments can be supported: we currently support a simple 'embedded' cloud that scales horizontally across a limited number of nodes and are experimenting with large-scale clustered environments that support load-balancing and fail-over.

8 Conclusion and Future Work

We have presented a method for enabling social participation around extension activities, workflow models and instances and an implementation supporting it in an underlying BPM as a service system. Our method and prototype offer a browser-based workflow editor, IDE support for extension activity publishing and reuse, a shared catalog of extension activities that supports community features, and a lightweight flow language and engine that support dynamic extensions, direct deployment, and browser-based interaction. Using this as a base, one could provide more advanced social software for hosted workflows such as community-based process improvement and a marketplace for extensions and workflows. We are currently working to address the equally important 'soft' issues (security, trust, IP in reuse, pricing, etc) in providing BPM as a service in the presence of the presented dynamic extensibility.

Acknowledgements. This work is part of a larger effort. A list of contributors is at `http://bit.ly/v1nqC`.

References

1. Armbrust, M., Fox, A., et al.: Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley (2009)
2. Benkler, Y.: The Wealth of Networks: How Social Production Transforms Markets and Freedom. Yale University Press (2006)
3. Benslimane, D., Dustdar, S., Sheth, A. (eds.): IEEE Internet Computing, special issue on Services Mashups, vol. 12. IEEE, Los Alamitos (2009)
4. Black Duck Software. Koders, `http://www.koders.com/`
5. Curbera, F., Duftler, M.J., Khalaf, R., Lovell, D.: Bite: Workflow composition for the web. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 94–106. Springer, Heidelberg (2007)
6. Desai, N., Mazzoleni, P., Tai, S.: Service Communities: A Structuring Mechanism for Service-Oriented Business Ecosystems. In: Digital EcoSystems and Technologies Conference (DEST 2007) (2007)
7. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, CA (2000)
8. Erol, S., Neumann, G.: From a social wiki to a social workflow system. In: BPM 2008 Workshops. LNBIB. Springer, Heidelberg (2008)
9. Geyer, W., Dugan, C., DiMicco, J., Millen, D.R., Brownholtz, B., Muller, M.: Use and reuse of shared lists as a social content type. In: CHI 2008, Florence, Italy. ACM, New York (2008)
10. GitHub.com. GitHub - Social Coding, `http://github.com/`
11. IBM. WebSphere sMash, `http://www.ibm.com/software/webservers/smash/`
12. Koschmider, A., Song, M., Reijers, H.A.: Social software for modeling business processes. In: BPM 2008 Workshops. LNBIB. Springer, Heidelberg (2008)
13. Lau, C.: BPM 2.0-a REST based architecture for next generation workflow management. In: Devoxx Conference, Antwerp, Belgium (2008), `http://www.devoxx.com/download/attachments/1705921/D8_C_11_07_04.pdf`
14. Mozilla Foundation. Bespin - Code in the Cloud, `https://bespin.mozilla.com/`
15. Paoli, H., Schmidt, A., Lockemann, P.C.: User-driven semantic wiki-based business service description. In: Int'l Conference on Semantic Technologies (I-Semantics 2007), Graz (2007)
16. ProgrammableWeb.com. Programmable Web, `http://www.programmableweb.com`
17. Rosenberg, F., Curbera, F., Duftler, M.J., Khalaf, R.: Composing RESTful services and collaborative workflows: A lightweight approach. IEEE Internet Computing 12(5) (2008)
18. Schmidt, R., Nurcan, S.: BPM and social software. In: Ardagna, D., et al. (eds.) BPM 2008. LNBIP, vol. 17, pp. 649–658. Springer, Heidelberg (2009)
19. Silva-Lepe, I., Subramanian, R., Rouvellou, I., Mikalsen, T., Diament, J., Iyengar, A.: SOAlive Service Catalog: A Simplified Approach to Describing, Discovering and Composing Situational Enterprise Services. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 422–437. Springer, Heidelberg (2008)
20. SourceForge, Inc. SourceForge.net, `http://www.sourceforge.net/`
21. Tai, S., Desai, N., Mazzoleni, P.: Service communities: Applications and middleware. In: Proc. of the Int'l Workshop on Software Engineering and Middleware (SEM 2006). ACM, New York (2006)
22. The Apache Ant Project. Ivy, `http://ant.apache.org/ivy/`
23. The Eclipse Foundation. Eclipse, `http://www.eclipse.org/`
24. Weerawarana, S., Curbera, F., Leymann, F., Ferguson, D.: Web Services Platform Architecture. Pearson Education, London (2005)