2012 ASE/IEEE International Conference on Social Computing and 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust

User Generated Human Computation Applications

Nadin Kokciyan
Department of
Computer Engineering
Bogazici University
Istanbul, Turkey
Email: nadin.kokciyan@boun.edu.tr

S. Uskudarli
Department of
Computer Engineering
Bogazici University
Istanbul, Turkey
Email: uskudarli@cmpe.boun.edu.tr

T. B. Dinesh
Servelots Infotech
Bangalore, India
Email: dinesh@servelots.com

***Abstract*—Social Web applications have successfully transformed content consuming users into content producers. Aside of socializing, these applications are frequently used to disseminate information and coordinate purposeful activities, such as disaster response, political action, and neighborhood organizations. These activities are carried out via human interpreted messages. In many cases, dedicated information processing spaces would better serve such needs. Unfortunately, a typical Web user is not able to create even very simple applications, as they require significant technical know-how. This work proposes an approach towards creating simple collaborative applications. The approach, called WeFlow, proposes a collaborative application specification, an application generator, and an execution engine. The WeFlow framework is presented with focus on the implementation issues. A case study, where users report and track accessibility violations, is presented for demonstration purposes.**

***Index Terms*—Human-centered computing, Collaborative computing, Human information processing, Workflow management, Computer-supported collaborative work**

I. Introduction

Collaborative applications such as wikis, social networking applications like Facebook and Twitter have greatly increased user participation. These applications are used to socialize, communicate, access or disseminate information, coordinate activities, etc. In fact, they are also utilized to serve purposeful activities, such as responding to disasters, coordinating political action, and organizing neighborhood information and activities. However, these platforms serve message transmission among humans who interpret and act on them. Whereas, a platform for *community information processing* would serve such needs much better, particularly in the long term. This is very unfortunate since the Web is such a great platform for collaboration. Many collaborative applications, such as Wikipedia and human computation applications [1], rely on user participation. The impressive results of such applications demonstrate the effectiveness of utilizing humans in their computation cycle. Participants are often unknown volunteers. Their contributions may be large or small. Most often the latter. Sometimes different kinds of contributions are performed, other times the same type of action is repeated by different participants, e.g. tagging pictures. While most users are able to generate content, they are not able to introduce new functionality. As a result they are not able to leverage the power of the Web to serve their needs.

The aim of this work is to address this gap in user defined Web applications. In another words, the idea is to empower typical Web users (who use the Web but can't program) with the ability to generate Web behavior (applications). This work introduces an approach towards a framework for creating user generated applications. A model (called WeFlow) that includes the specification, generation, and execution of participatory applications is presented. This approach is mainly influenced by *workflows* and *human computation*. This paper focuses on the implementation of the generation and execution of user defined participatory applications. A case study for reporting and tracking accessibility violations is presented for demonstrative purposes.

II. Background

Human computation applications involve many users making small contributions in order to accomplish a collaborative activity. A collaborative activity is broken down into tasks that are distributed among humans and computers based on their suitability for performing those tasks [2], [3]. Humans solve large-scale computation problems through online games, *Games with a Purpose* [4]. The reCAPTCHA [5] is a service that helps digitizing text (books, newpapers). An overview of human computation systems is detailed in Quinn and Bederson work [1].

A workflow consists of a sequence of small activities. Workflow participants (humans, computers) interact with workflows via workflow management (WFM) system. A WFM system defines, manages and executes workflows in a computer driven way. WFM systems have certain common characteristics [6], [7]: (1) modeling the workflow process and its activities, (2) managing the worklow process and sequencing activities, and (3) interacting with human users and IT applications for processing activities.

III. WeFlow: An Approach for Creating Human Computation Applications

WeFlow is about supporting collaborative purposeful activities. So, there is an overall activity that is to be carried out by a collection of people. The collection of people may be known in advance (neighbors) or may be unknown (anyone can volunteer). The basic idea is to partition the overall work into achievable tasks, make them available to participants, and coordinate the tasks until the work is completed. Of course, it is possible that some kind of work doesn't have a natural completion, such as defining an online encyclopedia.

A participatory human computation application specification (WeFlow Specification) consists of defining the following: (1) decomposing a collaborative activity into **tasks**, (2) describing the **resources** (participants) that can perform the tasks (human or machine), (3) defining the **control flow** which specifies sequencing constraints among tasks and (4) defining the **data flow** between tasks.

What the application designer is effectively doing is explicitly encoding the activity of coordinating collective tasks. Instead of making a to do list and keeping track of who is doing what towards a shared goal, this coordination is encoded in a Web application. Doing so, of course, expands the potential resources immensely, as potentially anyone on the Web can participate. The WeFlow framework can be viewed as a community information management application generator. WeFlow is based on a workflow model to handle data and control dependencies among tasks. Three main aspects are addressed:

1) **Collaborative Work Specification** in terms of the tasks, the dependencies among tasks, and resources that will perform the tasks.
2) **Generation** of Web applications corresponding to specifications.
3) **Execution** of generated Web applications.

WeFlow framework is depicted in Figure 1. Specifier is a regular Web user who is interested in creating a collaborative web application. Specifier defines a WeFlow Specification. WeFlow Application Generator generates a WeFlow Application (human computation web application) given a WeFlow Specification. WeFlow Execution Engine executes a WeFlow Application by presenting tasks to participants in accordance with the specification.

IV. WeFlow Specification

WeFlow Specifications describe collaborative activities. Creating a specification entails describing the tasks, people, control flow and data flow among tasks. Essentially this is a workflow definition for a distributed set of potentially unknown persons. The details of these aspects are provided in the following sections. WeFlow Specification is briefly described in this section. For further details, please refer to Chapter IV of [8].

A. Task

A task is a unit of a collaborative activity to be accomplished. WeFlow considers two categories of tasks based on who performs the task: *human task* and *automated task*. Each task that requires human feedback is considered a *Human Task*. An *Automated Task* refers to a computerized activity, Automated tasks may be performed by local systems or external systems (e.g. web services). As the focus of this work is related to human computation, we emphasize human tasks. Each task consists of:

1) **Inputs** A task may have several inputs that come from data created by prior tasks or from the person performing the task. There are two types of inputs that are typed variables:
   - zero or more *task input variables*, which receive values from previously performed tasks.
   - zero or more *human input variables*, which receive values from a human performing the task.
2) **Human Instruction** Human tasks require instructing the person what is expected from them. Any combination of the following may be used to specify these instructions:
   - text, e.g. "Tag this picture"
   - task input variables, e.g. "Upload *$cnt $animal_type* pictures".
   - human input variables, e.g. for the "Tag a Bird Picture" task, the instruction "Provide a tag *$tag* for this picture *$picture*", where *$picture* is a task input and *$tag* is a human input (*$* sign is a special character indicating a variable).
3) **Outputs** A task may have zero or more output variables which are typed variables. The value of an output variable is expressed in terms of task and human input variables.

Task Types: WeFlow provides some predefined types of tasks:

a) **Atomic Task:** is a basic task that cannot be broken down into other tasks.

b) **Conditional Task:** Task execution depends on a condition. A *Conditional Task* requires the specification of a *Conditional Expression* to satisfy and choose an appropriate task to execute.

IfElse Task provides a choice between two different tasks based on the satisfiability of the *Conditional Expression*. For example, consider a "Tag Something" task, where a person is asked to choose between 'audio' and 'image'. If the user chooses 'audio' then "Tag a Sound" task will be executed, otherwise "Tag an Image" will be executed. The output of IfElse task consists of the output of executed task.

Choice Task provides a choice between multiple different tasks. The task performer is asked to make a choice among a list of tasks. For example, a "What do you want to do?" task may present the user with a list of possible tasks to perform. The user simply makes a choice among the tasks, which will be the next executed task. The output of choice task consists of the output of executed task.

c) **Repetition Task:** computes a subtask several times in sequence. A repetition task consists of a *Conditional Expression* and a *Task*, its subtask. The *Task* is repeatedly executed as long as the *Conditional Expression* evaluates to *true*. Upon each completion of a *Task* the *Conditional Expression* is updated with the outputs of the *Task*. The output of repetition task consists of *Task*'s output. For example, the "Collect 50 Books" task defines a task for collecting books from people. The condition is such that if total number of books is less than 50, then this task will be repeated. Upon each completion of this task total number of books is updated. Once 50 books are

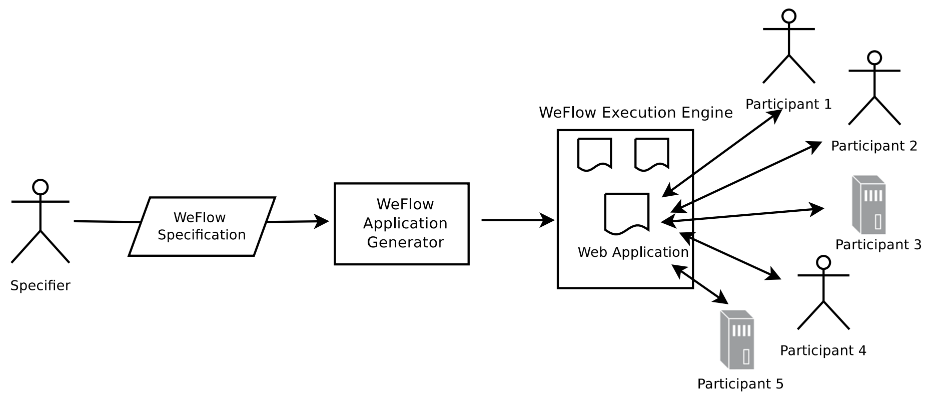


Fig. 1. Overview of The WeFlow Framework.

collected, then the repeat process terminates. If no restrictions are specified, each iteration may be performed by different users.

d) **DoAll Task:** groups a set of task (subtasks) that can be executed in parallel. These subtasks may be executed in any order. For example, the "Annotate a bird picture" task may be decomposed into the "Tag a bird picture" and "Identify the species of a bird" subtasks. These subtasks can be executed in parallel. The DoAll task is completed when all its subtasks are completed. The output of DoAll task consists of the output of its subtasks.

e) **Collective Task:** is defined by a subtask that should be executed multiple times in a non-sequential manner. The subtask is duplicated as many times as required. The result of a collective task is an aggregation of the results of each subtask. The subtask may be duplicated N number of times or set to infinite if the number of repetitions is unbounded. For example, a Collective Task "Tag a bird picture" with 5 times, defines a task for tagging a bird five times. The output of the collective task consists of an output array including five tags provided by human performers.

B. People

Once a collaborative activity is decomposed into tasks, it is also important to describe who can perform the tasks. It all depends on who is specifying the flow and who is interested in making use of it. The specification must make clear the policy of who can perform which tasks and what access rights they have. This is done with *role*s. In WeFlow Framework, workflow participants have roles and each human task is associated with one or more roles. This approach enables the classification of users, which can be utilized for describing permissions to a set of users rather than an individual approach. Thus, each human task may be performed by one of specified roles. WeFlow aims to generate human computation type web applications, where unknown users can participate. Tasks can not be assigned to specific users. Tasks are associated with roles, and roles are associated with people. A WeFlow Specification can only specify which tasks a user may perform indirectly through roles.

C. Flow

In a workflow, tasks have to be executed in a specific order and this sequencing in which work needs to be done is the *control flow* aspect of a workflow. At execution time, a task is initialized in terms of its inputs, i.e., data required to do the work has to be mapped to task input variables. A task is finalized in terms of its outputs, i.e., produced data has to be mapped to task output variables. This data perspective defines *data flow* aspect of a workflow. In WeFlow Specification, data flow is a declarative definition of data mappings between tasks and/or within tasks. According to data mapping information, a task is instantiated and executed. There are three types of data mappings:

1) Input Mappings: These are values mapped to inputs of task. These values come from previously performed tasks.

2) Human Input Mappings: These are data mappings between a human task and a human. At execution time of a task, data provided by human performer is mapped to human input variables. Communication between human tasks and humans is done through channels.

3) Output Mappings: An output variable value of a task is formulated in terms of input and human input variables of this task. Input variable values may directly be mapped to output variable values.

V. WEFLOW APPLICATION GENERATOR

A WeFlow Specification includes the specification of many tasks, and for each human task a Web page is generated by WeFlow Application Generator. Humans interact with generated Web application to perform tasks. Hence, a Web controller is also generated in order to handle human activities.

First, a task is represented on Web. Task instructions and task variables are mapped to HTML elements. For example, WeFlow *Text* datatype is represented as *<input type="text">* in HTML language. Second, human performer is informed by the task's human instruction. A task instruction is represented on Web as the following: (1) text is displayed as it is, (2) each referenced task input variable is replaced by its value, and (3) each referenced human input variable is replaced by a corresponding Web input method as described in the previous section.

VI. WeFlow Application Execution

WeFlow Execution Engine is the core of WeFlow framework. Web applications generated by WeFlow Application Generator are executed by this engine. It handles: (1) instantiation of Web applications, (2) instantiation of human tasks, (3) state of human participants and the workflow, and (4) distribution of tasks to workflow participants.

VII. A Human Computation Web Application

Banu is a student living in Rumelihisarustu and she's a regular Web user. She would like to create a place where people can: (1) report places not accessible in Rumelihisarustu such crumbling sidewalks, and/or objects in a bad condition such broken chairs, (2) fix reported problems, (3) verify whether a reported problem is fixed or not. Some of them work in reporting process, some in fixing process and some in verifying process. This application will be referred as Rumelihisarustu Accessibility project in the following sections.

A. Rumelihisarustu Accessibility Specification

Banu specifies Rumelihisarustu Accessibility in terms of WeFlow tasks, people, control and data flows among tasks, as depicted in Figure 2. Currently, this specification is given in XML format. Each rectangle represents a WeFlow task, each arrow shows a control flow between tasks, human tasks are denoted with a circle figure on top of the task and different colors show different groups. For clarity, data flow information is not depicted.

1) **Task Specification:** Each human task has a human instruction specified by Banu. For clarity, task instructions are not included here. Banu describes tasks as the following:

- *Choose Task* is a *Choice* task and allows a human to select one of two tasks: *Share Info* and *List Items*.
- *Share Info* is a human task. Human performer provides a picture url, information regarding this picture and a status flag, 'pending' or 'resolved'.
- *List Items* is an automated task and collects items provided by human performers from the data store.
- *Show Items* is a human task. Human performer sees a list of items and pick one item from the list by providing its id.
- *Pick Item* is an automated task. It gets an item from the data store given an item id.
- *Update Item* is a human task. Human performer sees a picture, some information and a status flag of an existing item. She updates status flag of this item.
- *Is Resolved* is an *IfElse* task. If status flag of an item is 'resolved' then *Verify Item* task is performed, *Choose Task* otherwise.
- *Verify Item* is a human task. Human performer verifies whether a reported item is really fixed or not. And she provides a status flag ('resolved' or 'pending') for this item.
- *Save Item* is an automated task. It stores an item to the data store.
- *Show Item* is a human task. Human performer sees a picture, some information and a status flag of an existing item.

2) **People Specification:** Banu specified tasks to be performed: *Choose Task*, *Share Info*, *Show Items*, *Update Item* and *Show Item* are human tasks to be performed by *anyone*; on the other hand, *Verify Item* task can only be performed by *verifiers group* users.

3) **Flow Specification:**

a) Control Flow Specification: In Figure 2, arrows depict control flow aspect in Rumelihisarustu Accessibility. For example, when the execution of *List Items* task is done, a human will perform *Show Items* task.

b) Data Flow Specification: After execution of a task, some data is provided by its performer and this data moves through the workflow. For example, *Pick Item* provides *picture* information which is mapped to *Update Item* input information. Input information may directly mapped to output information of a task. For example, *newstatus* is an human input information for *Update Item*. It is mapped to *Update Item* output information. All data mappings are done in a same way.

B. Rumelihisarustu Web Application Generation

An application place is created to store application specific files. For each human task, templates (web pages) are created. A controller is generated in order to handle web requests for Rumelihisarustu Accessibility.

C. Rumelihisarustu Web Application Execution

Banu uploads Rumelihisarustu Accessibility specification to WeFlow framework and starts the execution of this application by pressing *Start Workflow* button. Merve, who also lives in Rumelihisarustu, has some information to report. And she decides to join this application and she comes up with *Choose Task* task. After reading the task instruction, she selects *Share Info* task to perform. Merve provides some data: her picture url showing a broken sidewalk, some textual information related to this picture and status flag of the shared item which is 'pending'. Other people, living in the neighborhood, also report new items using the application. Hakan joins this application and selects *List Items* task in order to see reported items. He comes up with *Show Items* task where he finds two reported items. Hakan picks one item by providing its item id and wants to update information about an item previously reported by Merve, see Figure 3. And he modifies item's status flag to 'resolved'. Banu, who is the specifier of the application, is part of *verifiers group*. Thus, she may verify updated items rather they are fixed or not. As Hakan updated an item, Banu checks whether provided information is correct, see *Verify Item* task. And she decides that this is correct and types 'resolved' as the item status. Kaan joins the application and sees current list of reported items. Note that the status flag of the first item (uploaded by Merve) is updated to 'resolved' as verified by Banu.

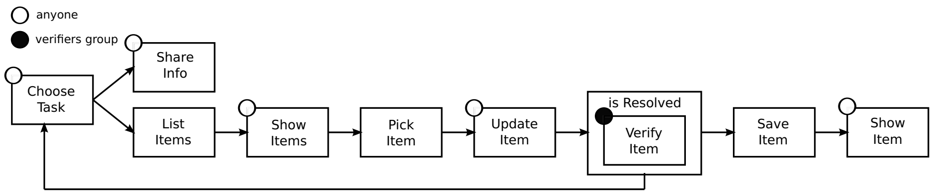
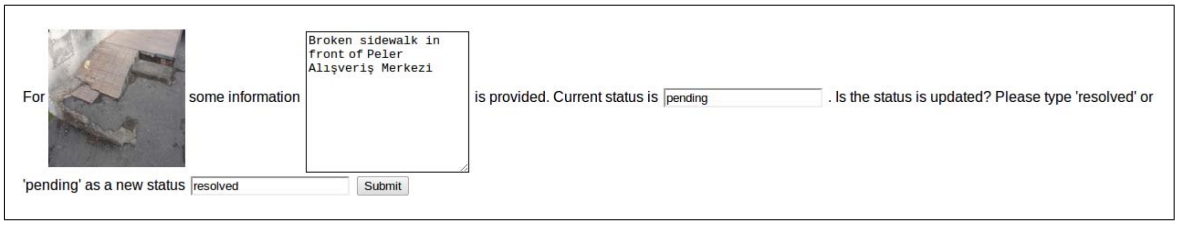


Fig. 2. Rumelihisarustu Accessibility Specification



For some information

```
Broken sidewalk in
front of Peler
Alışveriş Merkezi
```

is provided. Current status is pending . Is the status is updated? Please type 'resolved' or 'pending' as a new status resolved Submit

Fig. 3. Update Item

VIII. Implementation

Thus far, we have a preliminary implementation of the proposed approach which includes basic concepts of WeFlow framework. In this implementation, *Python* is used as the programming language, *ZODB* is chosen to be the data storage, and *web.py* the web server. Further details about the implementation can be found in Chapter V of [8].

First, a human computation web application is specified in XML. Second, a web application is generated given this XML specification. Screen generation captures the essential elements. Third, generated web application is executed. The WeFlow editor (user interface) to specify workflows has not yet been implemented. There has been no attempt to create appealing user interface designs, which obviously must be handled in the future.

A. WeFlow Specification Handler

WeFlow Specification is based on XML which is a standard format to keep data in a structured way. Tasks, people, control and data flow elements are described with specific XML structures. Further implementation details of this specification will not be detailed in this paper.

B. WeFlow Data Handler

ZODB is a native object database for Python and is widely used by Python community. For data storing purposes, ZODB is used as part of data layer of WeFlow Framework. There are two data handlers: (1) Specification Data Handler is used to get and/or store data related to WeFlow Specification, (2) Application Data Handler is used to get and/or store data related to running human computation web applications.

C. WeFlow User Handler

The User Handler deals with registering new users, authenticating users and creating new sessions, checking user access rights, deleting and/or modifying users, getting user workflows and setting user access rights.

D. WeFlow Application Generator

The WeFlow Application Generator Module generates a human computation web application using WeFlow Specification. HTML Data Generator maps WeFlow Specification task input data to *web.py* form elements. Application Template Generator creates human computation web application folder structure (physical space to keep files), templates (web views of human tasks) and a controller (handler for web requests). This module works along with some other libraries: (1) *htmlDoc* library is used in order to create HTML templates given HTML tags, (2) *PyGen* library is used in order to generate python files including python code.

E. WeFlow Execution Module

The WeFlow Execution Module is the core module of The WeFlow Framework. It instantiates workflows, executes all web applications, distibutes tasks to various workflow participants and aggregate responses, keeps track of the state of every web application and human performer. Some of its modules are as the following:

- *State Handler* keeps track of state information of (i) users, (ii) web applications. Given a workflow instance, this module gives current state of a web application and a user participant. State information is provided to Task Handler.
- *Task Handler* call tasks for execution. It gets next task information from *Control Flow Handler* module, checks type of next task and then call appropriate method. After execution, it calls State Handler module for updating state information of the running workflow instance.
- *Control Flow Handler* calls State Handler to check current state of an application instance. According to control flow semantics of the application instance, it decides next task to execute and then calls Task Handler module.
- *Instantiator* instantiates a web application once a new WeFlow specification is loaded to the WeFlow Frame-

work. This module also takes care of the instantiation of tasks in web applications.

F. WeFlow Tasklist Handler

This module manages the asynchronous interactions with workflow participants: list new tasks to users, provides required data to accomplish task and collect accomplished tasks via a web interface, as shown in Rumelihisarustu Accessibility application.

IX. Related Work

Facebook API provides facilities to develop web applications but its audience is IT-savvy people. Yahoo Pipes [9] enables users to create mashups using content on the Web. Many systems have been studied regarding micro-task markets such MTurk [10]. Crowdforge [11] is a framework that allows publishing tasks (textual descriptions) into micro-task markets. CrowdLang [12] is a programming language that allows publishing tasks into MTurk. Turkit [13] is a procedural programming language used to program iterative tasks in MTurk. CrowdWeaver [14] is a visual workflow management tool for crowd work. It does not provide a data model to describe tasks, i.e., it is not possible to collect, search and process data. It does not support to perform a task multiple times in a sequential/non-sequential way. Jabberwocky [15] is a social computing stack to write programs for crowdsourcing platforms. Unlike most of these systems require programming skills, *WeFlow* supports generation of web applications specified by regular Web users. Various workflow languages are designed including YAWL [16], [17] and SMAWL [18].

X. Discussion and Future Work

Communities use social web applications for their purposes. Current social web applications offer limited capabilities to keep data processable. Therefore, retaining the long term value of community information is becoming more crucial for communities of purpose as well. We intend to develop a visual language preserving WeFlow specification semantics, and let regular Web users specify their envisioned applications. WeFlow mobile version will simplify coordination. A robust data model will help communities to describe their domain better. We intend to add data export functionality for workflows: (1) allowing people to save their own data, (2) enabling data import/export from/to external systems such *Semantic Web*, *Linked Data*. We plan to create a library of predefined tasks and workflows to increase reusability. Furthermore, our initial model allows a task to be performed by a single individual at a time. This restriction will be relaxed to enable multiple people on performing a same task.

XI. Conclusions

We have developed a specification language to define human computation applications; an application generator that takes a human computation application specification and produces a web based application. The resulting web application can be used by people via a web browser. The execution engine executes the participatory web application by distributing the tasks to participants as defined by the application specification. We introduced one such a participatory web application Rumelihisarustu Accessibility. Combined with a richer data specification and a full fledged visual language, we envision a productive application specification environment.

Acknowledgment

The authors are grateful to Prof. L.G.L.T. Meertens and members of SoSLab (Complex Systems Lab at Bogazici University) for their feedback and contributions during this work. This work is partially funded by B.U. Research Fund (BAP5709) and TAM Project (2007K120610).

References

[1] A. J. Quinn and B. B. Bederson, “Human computation: a survey and taxonomy of a growing field,” in *Proceedings of the 2011 annual conference on Human factors in computing systems*, ser. CHI ’11. New York, NY, USA: ACM, 2011, pp. 1403–1412.

[2] L. von Ahn, “Human computation,” in *K-CAP ’07: Proceedings of the 4th international conference on Knowledge capture*. New York, NY, USA: ACM, 2007, pp. 5–6.

[3] T. W. Malone, R. Laubacher, and C. N. Dellarocas, “Harnessing Crowds: Mapping the Genome of Collective Intelligence,” *SSRN eLibrary*, 2009.

[4] L. v. Ahn, “Games with a purpose,” *Computer*, vol. 39, no. 6, pp. 92–94, Jun. 2006.

[5] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, “reCAPTCHA: Human-Based Character Recognition via Web Security Measures,” *Science*, vol. 321, no. 5895, pp. 1465–1468, Sep. 2008.

[6] D. Hollingsworth, “Workflow management coalition - the workflow reference model,” Workflow Management Coalition, Tech. Rep., Jan. 1995.

[7] R. Allen, “Workflow : An introduction,” *Production*, pp. 15–38, 1998.

[8] N. Kokciyan, “Weflow: We follow the flow,” Master’s thesis, Bogazici University, 2011.

[9] M. Pruett, *Yahoo! pipes*, 1st ed. O’Reilly, 2007.

[10] P. G. Ipeirotis, “Analyzing the amazon mechanical turk marketplace,” *XRDS*, vol. 17, no. 2, pp. 16–21, Dec. 2010.

[11] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, “Crowdforge: crowdsourcing complex work,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ser. UIST ’11. New York, NY, USA: ACM, 2011, pp. 43–52.

[12] P. Minder and A. Bernstein, “Crowdlang - first steps towards programmable human computers for general computation.” in *Human Computation*, ser. AAAI Workshops, vol. WS-11-11. AAAI, 2011.

[13] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, “Turkit: human computation algorithms on mechanical turk,” in *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, ser. UIST ’10. New York, NY, USA: ACM, 2010, pp. 57–66.

[14] A. Kittur, S. Khamkar, P. André, and R. Kraut, “Crowdweaver: visually managing complex crowd work,” in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, ser. CSCW ’12. New York, NY, USA: ACM, 2012, pp. 1033–1036.

[15] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar, “The jabberwocky programming environment for structured social computing,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ser. UIST ’11. New York, NY, USA: ACM, 2011, pp. 53–64.

[16] W. M. P. van der Aalst, “The application of petri nets to workflow management.” *Journal of Circuits, Systems, and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

[17] W. M. P. van der Aalst and A. H. M. ter Hofstede, “YAWL: yet another workflow language,” *Information Systems*, vol. 30, no. 4, pp. 245–275, Jun. 2005.

[18] C. Stefansen, “Smawl: A small workflow language based on ccs,” in *Harvard University*, 2005.