Workflow Definition by Cloud Collaboration

Conference Paper · January 2013

DOI: 10.4108/icst.collaboratecom.2013.254050

2 authors, including:

Gwan-Hwan Hwang
National Taiwan Normal University
58 PUBLICATIONS **333** CITATIONS

SEE PROFILE

Workflow Definition by Cloud Collaboration

Chi Wu-Lee and Gwan-Hwan Hwang
Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan
rekiwu@std.ntnu.edu.tw, ghhwang@csie.ntnu.edu.tw

***Abstract*—In this paper we propose a novel workflow definition language—called the CLWfDL (Cloud Collaboration Workflow Definition Language)—for defining workflow in the working model of cloud collaboration. The language enables the distributed definition and concurrent revision of a workflow by multiple users from different places in the cloud. Users who participate in defining a workflow can use the language to specify their own requirements for the workflow execution. Each user can selectively contribute to part of the workflow definition. Any conflicts between the requirements of different users can be detected automatically. A single flow-control construct such as AND-join or OR-split can incorporate the requirements of multiple users. We have also implemented a translator of the proposed language that can translate the requirements collected from multiple users into other workflow definition languages.**

Keywords-Workflow Management System; WfMS; Workflow definition language; Cloud; Cloud collaboration

I. INTRODUCTION

WfMSs (workflow management systems) are software systems that support coordination and cooperation among members of an organization when they are performing complex business processes [1]; these processes are modeled as workflow processes that are automated by the WfMS. The workflow model (also referred to as the *workflow process definition*) is the computerized representation of the business process that defines its starting conditions, stopping conditions, activities, and control and data flows among the involved activities. An activity is a logic step within a workflow that includes information about the starting and stopping conditions, the users who are allowed to participate (the *participants*), the tools and/or data needed to complete the activity, and the constraints on how the activity should be completed. The activities in a process are usually organized into a directed graph that defines the order of their execution, where nodes and edges in the graph represent activities and control flow, respectively. A *workflow process instance* represents the state of execution of a workflow process definition by the WfMS, and is usually controlled by the *workflow engine*.

A workflow process definition is usually presented in a *workflow definition (or description) language* that is used to express the causal or temporal dependencies among activities in the workflow process. There are several workflow definition languages, including XML (Extensible Markup Language) Process Definition Language (XPDL) [2], Business Process Execution Language (BPEL) [3], and "Yet Another Workflow Language" (YAWL) [4]. A workflow designer traditionally develops a workflow process definition using a graphical workflow editor, which provides an interface for defining and modifying workflows by arranging and connecting activities. It is possible to view and edit the workflow process definition, which the workflow editor stores in a workflow definition language. For example, the Enhydra JaWE is a graphical workflow editor that implements XPDL specification V2.1 using the Business Process Model and Notation (BPMN) graphical notation [5][6]; while the BPEL Designer project, a graphical editor, adds comprehensive support to Eclipse for the definition, authoring, editing, deploying, testing, and debugging of BPEL 2.0 [7]. Generally speaking, these graphical workflow editors consider a workflow as a directed graph that illustrates the execution sequence of the activities. Activities and flow-control constructs[1] are represented by the nodes, and transitions are represented by the edges in a directed graph.

Cloud collaboration is an emerging way of sharing and co-authoring documents through the use of cloud computing, whereby documents or objects are uploaded to central cloud servers, where they can be accessed by others (e.g., Google Docs and Google Calendar [8][9]). Cloud collaboration technologies allow users to upload and edit documents or objects within the cloud. Businesses are now increasingly switching to the use of cloud collaboration. New advances in cloud computing and collaboration are being prompted by the increasing need for firms to operate in an increasingly globalized world. Collaboration in this context refers to the ability of workers in a company to work together simultaneously on a particular task. With cloud collaboration users do not have to be in the same room or even the same country to effectively work together toward a common goal. In today's globalized society, having the capability to work together effectively is important to the productivity of an organization, and cloud collaboration tools are ideal for fostering this. In the past, most collaborations involving the production and editing of documents would have to be completed face to face. However, collaboration has become more complex, now involving working with people all over the

*The corresponding author is Gwan-Hwan Hwang (e-mail: ghhwang@csie.ntnu.edu.tw). C.Wu-Lee and G.H.Hwang's work was supported in part by Republic of China National Science Council under grant NSC 102-2219-E-003-002-.

[1] Common flow-control constructs of workflow process include "IF", "SEQUENCE", "WHILE", "REPEAT UNTIL", "AND-join", "OR-join", "AND-split", and "OR-split" [10].

978-1-936968-92-3 © 2013 ICST
DOI 10.4108/icst.collaboratecom.2013.254050

world in real time on a variety of different types of documents or objects, and using different devices.

This paper addresses the issue for defining the workflow process by cloud collaboration, which we show is not possible using a traditional workflow definition language. We propose a new workflow process definition language and framework targeting a collaborative definition of the workflow process. One of the main features of cloud collaboration is that multiple people work together simultaneously on a particular task. Consider the situation that multiple users from one or more organizations are working on defining a workflow process. Referring to Figure 1, the traditional way for n users (u_0, u_1, u_2, …,u_{n-1}) to define a workflow process collaboratively is for the users to specify their requirements [RQ(u_0), RQ(u_1), …, RQ(u_{n-1})] about the execution of a workflow process, which contain the proposed activities and the causal or temporal relationships between them in a workflow process. A workflow administrator then analyzes these requirements and employs a graphic workflow editor to make a workflow definition that will be saved in a specific workflow definition language. Finally, a workflow engine can read the workflow definition to control the execution of this workflow process. Sometimes the requirements [RQ(u_0), RQ(u_1), …, RQ(u_{n-1})] may *conflict* with each other. For example, RQ(u_i) contains the requirement that activity E can be started only when both activities A and B are finished and RQ(u_j) contains the requirement that activity E should be started immediately when either activity C or D is finished. The workflow administrator should report all the conflicts to users after performing the system analysis in order for users to revise their requirements so as to remove these conflicts. If multiple users revise an existing workflow definition, the workflow administrator should read the revised requirements and then modify the workflow definition in a graphical workflow editor. Users are generally not allowed to edit an existing definition directly, and especially not edit it concurrently.

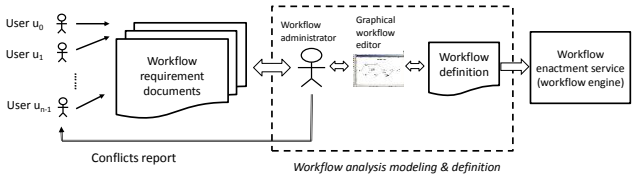

Figure 1. Traditional way to define a workflow process by multiple users

In the follows, we use a scenario to demonstrate that the traditional working model is inappropriate if we want to define a workflow process by cloud collaboration. Assume that there are four persons, Alice, Peter, John, and Bruce, who are chairmen from four communities in different universities. They are going to jointly organize a party for members in their communities. They want to define a workflow process which prepares this party so that they can use a WfMS to coordinate their works. Because their universities are located in different cities, they try to reduce the cost by defining a workflow process without meeting in the same location. They decide to employ the Google Docs to define this workflow process collaboratively. First, the four persons share a Google document. This document summarizes all the necessary activities in the workflow. They collaboratively edit this document. Assume that it results in a document as shown in Table 1. The workflow process has twelve activities.

TABLE 1. THE ACTIVITIES OF THE WORKFLOW

Activities	*Description*
Start	Workflow starts
End	Workflow ends
A1	Estimate the number of participants
A2	Book a place and decide the party time
A3	Reserve facilities and food
A4	Make Promotional materials
A5	Book a rain shelter
A6	Make and deliver propaganda posters
A7	Build a home page
A8	Participants make registration
A9	Book accessibility equipment
A10	Reserve food for vegetarians

Second, Alice, Peter, John, and Bruce need to define the causal or temporal relationships between these activities. To avoid the concurrent revision problem, they decide to use the *form tool* in Google Docs to collect their workflow requirements [11]. It is a tool to collect information quickly from multiple persons. Each person submits his own forms and the system will collect all the submitted forms and produce a report which is a table. The involved people can read the table which is composed of submitted forms before they submit their own forms. Assume that their submission of forms generates Table 2. Note that the submission time of a form and ID of the submitter are also shown in the table. Actually, Alice and Bruce submitted two forms.

TABLE 2. THE WORKFLOW REQUIREMENTS OF ALICE, PETER, JOHN, AND BRUCE

Timestamp	User	User Requirements
T1	Alice	A1 should be started after workflow starts.
		A1 should be finished before starting A2.
		A2 should be finished before starting A4.
		A4 should be finished before starting A6 and A8.
T2	Peter	A2 should be started after workflow starts.
		A2 should be finished before starting A3.
		A3 should be finished before starting A4.
T3	John	A8 should be finished before starting A9
		A9 should be finished before workflow ends.
T4	Bruce	If the probability of precipitation is bigger than 60% on the party date, A5 should be started after A2 finished.
		A4 should be finished before starting A7 and A8.
T5	Alice	A8 should be finished before starting A10.
		A10 should be finished before workflow ends.
T6	Bruce	A6 and A7 should be finished before workflow ends.

Finally, Alice, Peter, John, and Bruce need to hire a person to analyze the requirements shown in Table 2. That person plays a role of the workflow administrator. In the working model shown in Figure 1, all editing of the workflow definition should be performed by the workflow administrator. The workflow requirement in a natural language usually has to be analyzed by a human because no tool is available to automatically translate the numerous requirements into a workflow process definition. In this situation, whenever any user updates his or her workflow requirement or supplies a new requirement, the workflow definition cannot be amended automatically. The working model therefore does not fit the working model of cloud collaboration.

One of the solutions that addresses the inadequacy of the workflow model shown in Figure 1 is to give each user the privileges necessary to revise the workflow definition in a graphical workflow editor. Without loss of generality, we assume that multiple users are allowed to edit a directed graph simultaneously, because a workflow definition is usually modeled as a directed graph [2][3][4]. Consider a simple workflow definition of a workflow process that consists of three activities as shown in Figure 2A. In this example, user u_i is going to add an activity D after activity A, as shown in Figure 2B, and user u_j is going to remove activity B, as shown in Figure 2C. It is obvious that the final workflow process should be the one shown in Figure 2D. Users u_i and u_j can revise the workflow definition shown in Figure 2A concurrently. User u_i would employ a graphical workflow editor to perform the following operations: $OP_{i,1}$—remove edge (A, B), $OP_{i,2}$—add node D, $OP_{i,3}$—add edge (A, D), and $OP_{i,4}$—add edge (D, B); while user u_j issues the following operations: $OP_{j,1}$—remove edge (A, B), $OP_{j,2}$—remove edge (B, C), $OP_{j,3}$—remove node B, and $OP_{j,4}$—add edge (A, C). Since the two users are working concurrently, they read the same definition and their operations will interleave. It seems impossible to find a suitable interleaving that would allow their operations to execute properly. For example, if we execute operations in the order ($OP_{j,1}$, $OP_{j,2}$, $OP_{j,3}$, $OP_{j,4}$, $OP_{i,1}$, $OP_{i,2}$, $OP_{i,3}$, $OP_{i,4}$), operations $OP_{i,1}$ and $OP_{i,4}$ cannot be executed because activity B does not exist after the execution of operations $OP_{j,1}$, $OP_{j,2}$, $OP_{j,3}$, and $OP_{j,4}$. This kind of *concurrent editing error* occurs when multiple users are trying to alter specific data items simultaneously [12]. It can be prevented by the restriction of allowing only a single user to edit the workflow definition at a time, which can be implemented by a lock mechanism [13]. However, employing a lock mechanism to prevent concurrent editing has two drawbacks: (1) it may result in inefficiency, since a user might lock a workflow definition for editing for too long, resulting in a bottleneck; and (2) there is no mechanism to prevent a user from modifying the requirements of other users.

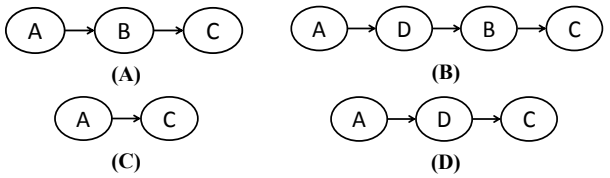

Figure 2. A sample workflow definition

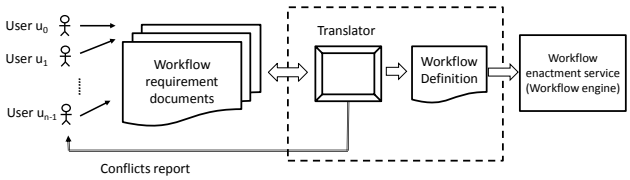

Figure 3. Cloud collaboration of the workflow definition

This paper proposes a working model for cloud collaboration in workflow definition. Referring to Figure 3, consider n users (u_0, u_1, u_2, …, u_{n-1}) who are wanting to define a workflow process collaboratively. These users specify their requirements for workflow executions [$RQ(u_0)$, $RQ(u_1)$, …, $RQ(u_{n-1})$], and a *translator*, which could be a computer program, reads these requirements and translates them into a workflow definition that defines an appropriate workflow process. A human is not needed to analyze the workflow requirements, and each user contributes to part of the workflow definition. Whenever a user u_i revises his/her own requirement, $RQ(u_i)$, the translator updates the workflow definition accordingly. It is possible that multiple users will submit their requirements concurrently, in which case the translator can still generate the revised workflow definition as long as there is no conflict within $RQ(u_0)$, $RQ(u_1)$, …, and $RQ(u_{n-1})$. In cases where there is a conflict, the translator informs the involved users immediately, who then need to revise their requirements and submit them accordingly. This process is repeated until an agreed workflow definition is finally obtained from multiple users.

We propose a workflow definition language to support collaborative workflow definition in the cloud. Consider n users (u_0, u_1, u_2, …, u_{n-1}), where each user u_i ($0 \le i < n$) specifies his/her workflow requirement $RQ(u_i)$ distributively. $RQ(u_i)$ ($0 \le i < n$) is part of the definition of a workflow process. A translator can read all the requirements [$RQ(u_0)$, $RQ(u_1)$, …, $RQ(u_{n-1})$] to construct a complete workflow process definition. The translator can also determine if there is any conflict between these workflow requirements. Users are allowed to revise their workflow requirement concurrently, but they are not able to modify the requirements of other users. A human does not need to analyze the workflow requirements.

This paper is organized as follows. Section II proposes a novel definition language that supports workflow definition by cloud collaboration. Section III discusses how to deal with the situation when there is conflict among the requirements of different users. Section IV summarizes the capability for distributed definition and concurrent revision provided by the proposed definition language. Section V presents an API to support the implementation of the translator shown in Figure 3 based on the proposed definition language. Section VI surveys previous work, and conclusions are drawn in Section VII.

II. A Definition Language For Collaboration Workflow Definition

This section presents a workflow definition language that fits the working model of cloud collaboration, which is called the CLWfDL (Cloud Collaboration Workflow Definition Language). As we have mentioned, each user u_i ($0 \le i < n$) specifies his/her workflow requirement $RQ(u_i)$ distributively. In the CLWfDL, $RQ(u_i)$ consists of a set of rules, where each rule γ is either ($x \succ_{\{\mu, f(z)\}} y$) or ($x \prec_{\{\mu, f(z)\}} y$), defined as follows:

- "x" and "y" are activities in the defined workflow process, which are called the *antecedent* and *consequent* activities of γ, respectively.
- "μ" specifies the relationship between "x" and "y". f(z) is a Boolean predicate, where "z" represents the set of parameters for evaluating the predicate. f(z) can be absent, in which case the rule is either ($x \succ_{\mu} y$) or ($x \prec_{\mu} y$).
- "$\succ$" and "$\prec$" are used to specify the type of "μ", being *join* and *split*, respectively.

We now describe the abstract meaning of the two different types of rules. First, we consider using the CLWfDL to define the AND-join relationship between multiple activities. Referring to Figure 4, the rule ($x \succ_{\text{AND-join}} y$) means that activity x is one of the activities that enables the AND-join transition to make activity y start (Figure 4A), while ($x \succ_{\text{OR-join}} y$) means that activity x is one of the activities that enables the OR-join transition to make activity y start (Figure 4B). Similarly, ($x \prec_{\text{OR-split}} y$) indicates an OR-split construct (Figure 4C). These three types of rules can be associated with a Boolean predicate, as shown in Figure 4D, E, and F.

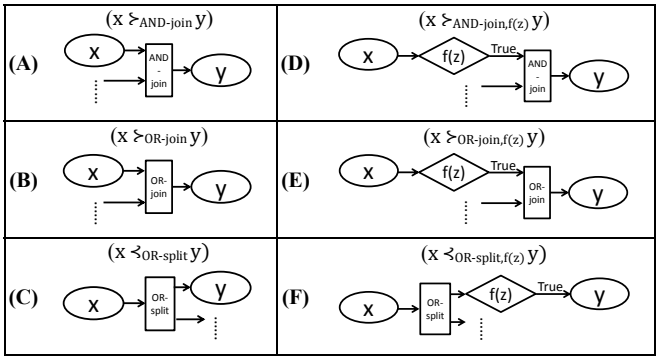

Figure 4. Example CLWfDL rules

In the CLWfDL, a single flow-control construct such as AND-join, OR-join, or OR-split can comprise rules from different users. For example, consider the situation where the requirements of Alice, John, and Bruce are RQ(Alice) = {…, A10 $\succ_{\text{AND-join}}$ End, …}, RQ(John) = { …, A9 $\succ_{\text{AND-join}}$ End,…}, and RQ(Bruce) = { …, A6 $\succ_{\text{AND-join}}$ End, A7 $\succ_{\text{AND-join}}$ End,…}, respectively. This requires the workflow definition to contain the AND-join flow-control construct involving activities A6, A7, A9, A10, and End, as shown in Figure 5. This construct means that activities A6, A7, A9, and A10 have finished if and only if End has started. Also, we can employ the AND-join construct to define a sequence relationship between activities. For example, the two rules (A $\succ_{\text{AND-join}}$ B) and (B $\succ_{\text{AND-join}}$ C) define the sequence relationship of A, B, and C as shown in Figure 2A.

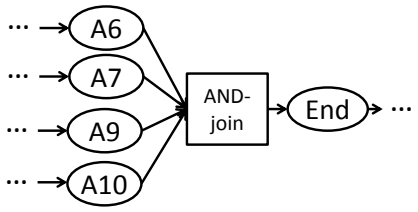

Figure 5. An AND-join construct

To show the defining power of CLWfDL rules, we first demonstrate how to use them to define all the flow-control constructs defined in BPEL that contain <sequence>, <if>, <while>, <repeatUntil>, <flow>, <pick>, sequential <forEach>, and parallel <forEach>. In [14], Appendix A shows illustrative directed graphs and the corresponding CLWfDL rules that implement these constructs. Appendix B discusses how to use CLWfDL rules to define other advanced flow-control constructs.

We now discuss how to make a translator for CLWfDL rules as shown in Figure 3. Our idea is to design an algorithm that can translate a set of CLWfDL rules into a directed graph in which nodes are activities and flow-control constructs and edges are transitions. The workflow definitions for existing workflow description languages can then be generated according to the directed graph so that we can employ an existing workflow engine to provide the *workflow enactment service* (see Figures 1 and 2).

Consider n users (u_0, u_1, u_2, …,u_{n-1}), where each user u_i ($0 \le i < n$) specifies his/her workflow requirement $RQ(u_i)$. The workflow enactment service should satisfy $RQ(u_0)$, $RQ(u_1)$, …, and $RQ(u_{n-1})$. Assume that $RQ(u_i)$ consists of a set of CLWfDL rules. A system is needed that controls the execution of the workflow process according to rules in $RQ(u_0) \cup RQ(u_1) \cup \ldots \cup RQ(u_{n-1})$. We employ the following notations to explain our idea:

- Γ is the set of all rules from all users in the workflow definition; that is, $\Gamma = RQ(u_0) \cup RQ(u_1) \cup \ldots \cup RQ(u_{n-1})$.
- $\Phi_{a,\mu}(\Gamma)$ denotes a set of join rules in Γ that have the $\succ$ type of "μ" and an identical consequent activity "a". Thus, we have $\Phi_{a,\mu}(\Gamma) = \{ \gamma \mid \gamma = (x \succ_{\mu} a)$ or $\gamma = (x \succ_{\mu,f(z)} a), \gamma \in \Gamma \}$.
- $\Psi_{a,\mu}(\Gamma)$ denotes a set of split rules in Γ that have the $\prec$ type of "μ" and an identical antecedent activity "a". Therefore, we have $\Psi_{a,\mu}(\Gamma) = \{ \gamma \mid \gamma = (a \prec_{\mu} y)$ or $\gamma = (a \prec_{\mu,f(z)} y), \gamma \in \Gamma \}$.
- The set of antecedent activities of a set of rules $\mathbb{R}$, denoted $AAS(\mathbb{R})$, consists of the union of the antecedent activities of rules in $\mathbb{R}$. If the rule has a Boolean predicate, the activity is associated with this predicate. Thus, we have $AAS(\mathbb{R}) = \{ x \mid (x \succ_{\mu} a) \in \mathbb{R} \} \cup \{ x_{f(z)} \mid (x \succ_{\mu,f(z)} a) \in \mathbb{R} \} \cup \{ x \mid (x \prec_{\mu} a) \in \mathbb{R} \} \cup \{ x_{f(z)} \mid (x \prec_{\mu,f(z)} a) \in \mathbb{R} \}$.
- The set of consequent activities of a set of rules $\mathbb{R}$, denoted $CAS(\mathbb{R})$, consists of the union of the consequent activities of rules in $\mathbb{R}$. If the rule has a Boolean predicate, the activity is associated with this predicate. Thus, we have $CAS(\mathbb{R}) = \{ y \mid (a \succ_{\mu} y) \in \mathbb{R} \} \cup \{ y_{f(z)} \mid (a \succ_{\mu,f(z)} y) \in \mathbb{R} \} \cup \{ y \mid (a \prec_{\mu} y) \in \mathbb{R} \} \cup \{ y_{f(z)} \mid (a \prec_{\mu,f(z)} y) \in \mathbb{R} \}$.

For example, if we have Γ_1 = { Start $\succ_{\text{AND-join}}$ A1, A1 $\succ_{\text{AND-join}}$ A2, A2 $\succ_{\text{AND-join}}$ A4, A4 $\prec_{\text{AND-split}}$ A6, A4 $\prec_{\text{AND-split}}$ A8, A8 $\prec_{\text{AND-split}}$ A10, A10 $\succ_{\text{AND-join}}$ End, Start $\succ_{\text{AND-join}}$ A2, A2 $\prec_{\text{AND-split}}$ A3, A3 $\succ_{\text{AND-join}}$ A4, A8 $\prec_{\text{AND-split}}$ A9, A9 $\succ_{\text{AND-join}}$ End, A2 $\prec_{\text{AND-split,f(z)}}$ A5, A4 $\prec_{\text{AND-split}}$ A7, A6 $\succ_{\text{AND-join}}$ End, A7 $\succ_{\text{AND-join}}$ End }, then we have (1) $\Phi_{A4,\text{AND-join}}(\Gamma_1)$ = { A2 $\succ_{\text{AND-join}}$ A4, A3 $\succ_{\text{AND-join}}$ A4 }, (2) $\Psi_{\text{Start,AND-join}}(\Gamma_1)$ = { Start $\succ_{\text{AND-join}}$ A1, Start $\succ_{\text{AND-join}}$ A2 }, (3) $AAS(\Gamma_1)$ = { Start, A1, A2, $A2_{f(z)}$, A3, A4, A6, A7, A8, A9, A10 }, (4) $AAS(\Phi_{A4,\text{AND-join}}(\Gamma_1))$ = { A2, A3 }, (5) $CAS(\Gamma_1)$ = { A1, A2, A3, A4, $A5_{f(z)}$, A6, A7, A8, A9, A10, End }, and (6) $CAS(\Phi_{A4,\text{AND-join}}(\Gamma_1))$ = { A4 }.

Given $\Gamma = RQ(u_0) \cup RQ(u_1) \cup \ldots \cup RQ(u_{n-1})$, the workflow enactment service should fit the following requirements according to rules in Γ. An activity derived from

AAS() or CAS() might or might not be associated with a Boolean predicate. We define an activity that is not associated with a Boolean function as being *finished* if its execution is finished, while an activity that is associated with a Boolean predicate (e.g., $B_{f(z)}$) is finished when its execution is finished and the associated Boolean predicate is True. Similarly, an activity that is associated with a Boolean predicate f(z) can be *started* only if f(z) is True. We define that a set $\mathbb{R}$ of rules is *satisfied*, denoted Satisfy($\mathbb{R}$), if the predicates P1 and P2 are True. Assume that Ω is the set of all activities that are referred in $\mathbb{R}$, then

- ***P1***: $\forall$ A in Ω and $\forall$ μ in $\mathbb{R}$, the execution of A and activities in AAS($\Phi_{A,\mu}(\mathbb{R})$) should be controlled according to μ.
- ***P2***: $\forall$ A in Ω and $\forall$ μ in $\mathbb{R}$, the execution of A and activities in CAS($\Psi_{A,\mu}(\mathbb{R})$) should be controlled according to μ.

For example, if μ is an AND-join construct, then all activities in AAS($\Phi_{A,AND\text{-}join}(\mathbb{R})$) are finished if and only if A has started; if μ is an OR-join construct, then one of the activities in AAS($\Phi_{A,OR\text{-}join}(\mathbb{R})$) is finished if and only if A has started; and if μ is an OR-split construct, then A is finished if and only if one of the activities in CAS($\Psi_{A,OR\text{-}split}(\mathbb{R})$) has started. Note that we assume that there is no conflict between rules—in section III we discuss how to deal with such conflicts. Predicates P1 and P2 define the causal or temporal relationship between activities from a set of rules. The fulfillment of Satisfy(Γ) obeys the causal or temporal relationship between activities defined by all the users. Algorithm 1 shows how to generate a directed graph from a set of CLWfDL rules, Γ. Actually, steps (4) and (5) ensure that the generated directed graph satisfy predicates P1 and P2, respectively. The generated directed graph G represents a workflow definition that fulfills Satisfy(Γ).

Algorithm 1:
Generate a directed graph from a set of CLWfDL rules.
Input: A set of CLWfDL rules, Γ.
Output: A directed graph, G.
GenerateDG(Γ)
(1) Construct a directed graph G = (V, E) with no node (V = $\varnothing$[2]) or edge (E = $\varnothing$).
(2) Let Ω be the set of all activities involved in Γ.
(3) For each activity in Ω, create an activity node that is named by the ID of the activity, and add this node to V.
(4) FOR EACH activity α in Ω and μ in Γ
// ***To satisfy predicate P1***
- IF $\Phi_{\alpha,\mu}(\Gamma) \neq \varnothing$ THEN
  - Add a node named "μ_α" to V.
  - Add edge (μ_α, α) to E.
  - FOR EACH activity β in AAS($\Phi_{\alpha,\mu}(\Gamma)$)
    - ◊ IF β is associated with a Boolean predicate f(z)
      - ♦ IF there is not a node "IF_fz" in V
        - ○ Add an IF-ELSE node named "IF_fz" to V.
      - ♦ Add edge (β, IF_fz) to E.
      - ♦ Add edge (IF_fz, μ_α) to E.
    - ◊ ELSE IF β is not associated with a Boolean predicate f(z)
      - ♦ Add edge (β, μ_α) to E.
  - END FOR

END FOR
(5) FOR EACH activity α in Ω and μ in Γ
// ***To satisfy predicate P2***
- IF $\Psi_{\alpha,\mu}(\Gamma) \neq \varnothing$ THEN
  - Add a node named "μ_α" to V.
  - Add edge (α, μ_α) to E.
  - FOR EACH activity β in CAS($\Psi_{\alpha,\mu}(\Gamma)$)
    - ◊ IF β is associated with a Boolean predicate f(z)
      - ♦ IF there is not a node "IF_fz" in V
        - ○ Add an IF-ELSE node named "IF_fz" to V.
      - ♦ Add edge (IF_fz, β) to E.
      - ♦ Add edge (μ_α, IF_fz) to E.
    - ◊ ELSE IF β is not associated with a Boolean predicate f(z)
      - ♦ Add edge (μ_α, β) to E.
  - END FOR

END FOR
(6) Remove all the unnecessary control nodes, which are nodes with only one in-edge and out-edge in V.
(7) Return G.

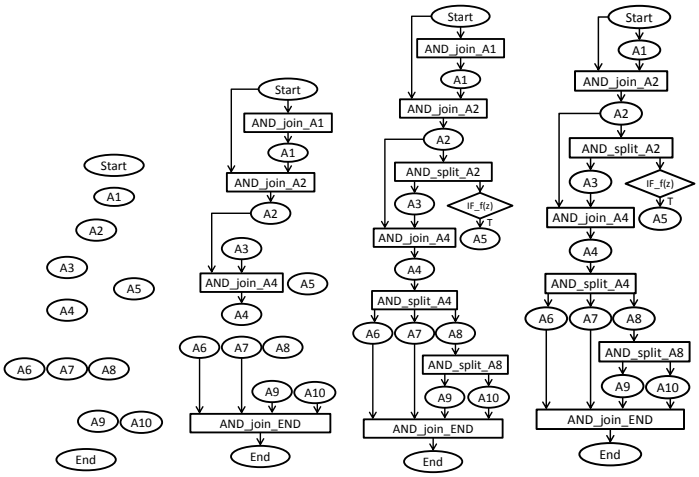

Figure 6. Illustration of Algorithm 1

We now employ our motivating example to demonstrate Algorithm 1. Referring to Table 2, Alice, Peter, John, and Bruce can use CLWfDL to specify their requirements as follows:

- RQ(Alice) = { Start $\succ_{AND\text{-}join}$ A1, A1 $\succ_{AND\text{-}join}$ A2, A2 $\succ_{AND\text{-}join}$ A4, A4 $\prec_{AND\text{-}split}$ A6, A4 $\prec_{AND\text{-}split}$ A8, A8 $\prec_{AND\text{-}split}$ A10, A10 $\succ_{AND\text{-}join}$ End }.
- RQ(Peter) = { Start $\succ_{AND\text{-}join}$ A2, A2 $\prec_{AND\text{-}split}$ A3, A3 $\succ_{AND\text{-}join}$ A4 }.
- RQ(John) = { A8 $\prec_{AND\text{-}split}$ A9, A9 $\succ_{AND\text{-}join}$ End }.
- RQ(Bruce) = { A2 $\prec_{AND\text{-}split,f(z)}$ A5, A4 $\prec_{AND\text{-}split}$ A7, A4 $\prec_{AND\text{-}split}$ A8, A6 $\succ_{AND\text{-}join}$ End, A7 $\succ_{AND\text{-}join}$ End }, where f(z) is a Boolean predicate which returns True if the

[2] "$\varnothing$" is the empty set.

probability of precipitation is bigger than 60% on the party date.

In step (2) we have Ω = { Start, End, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10 }, and after step (3) we have G, as shown in Figure 6A. In step (4), we add some AND-join nodes to satisfy predicate P1, which results in Figure 6B. Figure 6C results from step (5) by adding some AND-split nodes to satisfy predicate P2. In step (6) we remove all the unnecessary control nodes (i.e., AND-join, or AND-split nodes that have only a single in-edge and out-edge), finally yielding G = (V,E) as shown in Figure 6D.

Note that we have two special activities, "Start" and "End", which indicate the start and end of the workflow process. Actually, the CLWfDL can also be used to define workflow exceptions [15]. For example, we can use the rule "ExceptionRaise(X) $\succ_{\text{OR-join}}$ A1" to define when a workflow exception X occurs and we have to execute A1.

III. Conflicts Between CLWfDL Rules

A workflow definition could contain some errors such as the existence of unreachable activities, deadlock, and a poorly well-defined structure [16]. The traditional way to handle errors in a workflow definition is to discover them before the workflow process is executed. The rules in the CLWfDL usually come from many people, including some who might not communicate, which increases the likelihood of conflicts between rules. For example, if r1 = (A $\succ_{\text{AND-join}}$ E), r2 = (B $\succ_{\text{AND-join}}$ E), r3 = (C $\succ_{\text{OR-join}}$ E), and r4 = (D $\succ_{\text{OR-join}}$ E), then there exists a conflict for Satisfy(Γ), where Γ contains r1, r2, r3, and r4. According to the semantics of the AND-join construct, activity E is started after both activities A and B are finished due to r1 and r2. According to the semantics of the OR-join construct, activity E should be started immediately when either C or D is finished. Thus, when activity C is finished and A is not finished, we cannot satisfy r1, r2, r3, and r4 at the same time. This situation is referred to as a rule conflict. It is possible to have different types of conflict between a set Γ of CLWfDL rules:

- If (a $\succ_{\mu 1,f(z)}$ x) ∈ Γ and (b $\succ_{\mu 2,f(z)}$ x) ∈ Γ, where a ≠ b and μ1≠μ2, then Satisfy(Γ) is not possible because μ1 and μ2 cannot be satisfied at the same time.
- If (a $\prec_{\mu 1,f(z)}$ x) ∈ Γ and (b $\prec_{\mu 2,f(z)}$ x) ∈ Γ, and a ≠ b and μ1≠μ2, then Satisfy(Γ) is not possible because μ1 and μ2 cannot be satisfied at the same time.
- If (a $\prec_{\mu 1,f(z)}$ x) ∈ Γ and (b $\succ_{\mu 2,f(z)}$ x) ∈ Γ, and a ≠ b and μ1≠μ2, then Satisfy(Γ) is not possible because μ1 and μ2 cannot be satisfied at the same time.

When there are conflicts in Γ, there is no way for the workflow enactment service to control the execution of activities so as to satisfy all the rules in Γ. An intuitive approach is that the system should report conflicts to users as shown in Figure 3. The users read the conflict report and revise their rules accordingly.

Another solution is to associate *priorities* with rules. The use of priority can alleviate the conflicts between rules. A rule with a priority is either $(x \succ_{\mu} a)_p$ or $(x \succ_{\mu,f(z)} a)_p$, where p is an integer number greater than or equal to zero. We abbreviate a rule with the lower priority "0" [e.g., $(x \succ_{\mu} a)_0$] as $x \succ_{\mu} a$. Referring to Algorithm 2, by removing rules that conflict with other rules and have a lower priority, we can derive a set of rules, Γ', that includes rules with a higher priority. Note that it is still possible that rules in Γ' conflict with each other, since rules with the same priority will not be removed by Algorithm 2. If such conflicting rules exist in Γ', we should inform the involved users.

Algorithm 2: Removing conflicts from a set of CLWfDL rules according to the rule priorities
Input: A set of CLWfDL rules, Γ.
Output: A set of CLWfDL rules, Γ'.
RemovingConflict(Γ)
(1) Let Ω be the set of all activities involved in Γ.
(2) FOR EACH rule pair (r_i, r_j) in Γ
- IF r_i and r_j are in conflict and Priority(r_i) < Priority(r_j) THEN
  - Γ' = Γ – { r_i }.

END FOR
Return Γ'.

Referring to Figure 6, if Peter changes his requirement to RQ(Peter)' = { Start $\succ_{\text{AND-join}}$ A2, A2 $\prec_{\text{AND-split}}$ A3, A3 $\succ_{\text{AND-join}}$ A4, A3 $\succ_{\text{AND-join}}$ A9 }. There is a conflict between RQ(Peter)', and RQ(John). Because (A3 $\succ_{\text{AND-join}}$ A9) and (A8 $\prec_{\text{AND-split}}$ A9) ∈ Γ, A3 ≠ A8, and AND-join ≠ AND-split. If Peter assigns priority one to his rule: $(A3 \succ_{\text{AND-join}} A9)_1$, than Priority(A3 $\succ_{\text{AND-join}}$ A9) is greater than Priority(A8 $\prec_{\text{AND-split}}$ A9). Thus we can get Γ' = Γ – { A8 $\prec_{\text{AND-split}}$ A9 } by Algorithm 2. We therefore input Γ' when executing Algorithm 1. The result is shown in Figure 7. If there is no conflict between rules in Γ' after the applying of Algorithm 2, we can create a directed graph G by applying Algorithm 1.

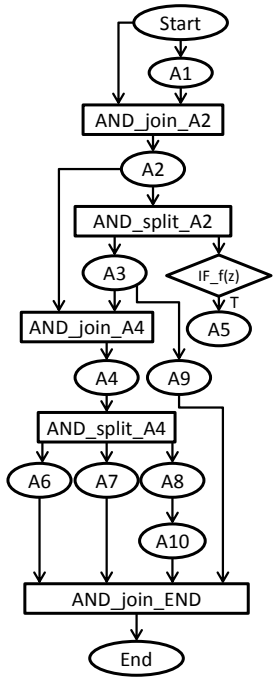

Figure 7. The result of applying Algorithm 1 to Γ'

IV. DISTRIBUTED DEFINITION AND CONCURRENT REVISION IN THE CLWFDL

In this section we summarize the definition and revision model that the CLWfDL can support. First, the CLWfDL supports a kind of *distributed definition* of workflow processes (see Theorem 1). Assuming that there are n users (u_0, u_1, u_2, …, and u_{n-1}), in this paper we say that these n users can distributively define a workflow process if there exists a program tool that can translate their requirements [$RQ(u_0)$, $RQ(u_1)$, …, $RQ(u_{n-1})$] into a representative form of a workflow definition that is aware of some workflow engine without assistance from humans, and can identify conflicts between the requirements of different users. Also, a single flow-control construct that involves multiple activities can be determined from the diverse requirements of multiple users.

Theorem 1: The CLWfDL can support a distributed definition of workflow processes.

Proof: *First, the CLWfDL supports rules for users to construct their requirements, and these rules can define all the possible flow-control structures as shown in [14]. Second, there exist algorithms to construct a directed graph from users' requirements and identify conflicts as shown in Algorithm 1 and section III, respectively. QED*

The CLWfDL supports a revision model that is appropriate for cloud collaboration, which we call *concurrent revision* (see Theorem 2). Assume that a workflow definition D is defined distributively by n users, which is denoted as $D = RQ(u_0) \cup RQ(u_1) \cup \ldots \cup RQ(u_{n-1})$. A concurrent revision of D to D' involves each user u_i ($0 \le i < n$,) revising his/her own requirement $RQ(u_i)$ independently and simultaneously without referring to the requirements of other users. If user u_i ($0 \le i < n$) revises $RQ(u_i)$ to $RQ'(u_i)$, then $D' = RQ'(u_0) \cup RQ'(u_1) \cup \ldots \cup RQ'(u_{n-1})$.

Theorem 2: The CLWfDL can support the concurrent revision of workflow processes.

Proof: *Since there exists an algorithm (i.e., Algorithm 1) that can generate a directed graph for a workflow, the requirements of users [i.e.,* $RQ(u_0)$, $RQ(u_1)$, …, $RQ(u_{n-1})$*] can be stored in their original form. The users simply revise their requirements concurrently and then submit them to Algorithm 1. Also, a user cannot modify the requirements of others. QED*

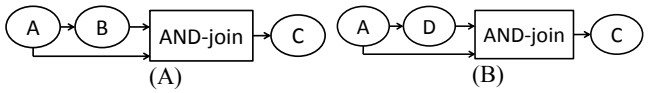

Figure 8. The directed graph generated by revised CLWfDL rules

We employ the example presented in Figure 2 to demonstrate how the CLWfDL rules support concurrent revision. Assume that two users, u_i and u_j, are making a workflow definition using CLWfDL rules. The original requirements are $RQ(u_i) = \{ A \succ_{AND\text{-}join} C \}$ and $RQ(u_j) = \{ A \succ_{AND\text{-}join} B, B \succ_{AND\text{-}join} C \}$. Submitting $RQ(u_i) \cup RQ(u_j)$ to Algorithm 1 yields the directed graph shown in Figure 8A. According to the scenario, user u_i is going to add an activity D after activity A, and user u_j is going to remove activity B. Therefore, u_i and u_j revise their original requirements to $RQ'(u_i) = \{ A \succ_{AND\text{-}join} D, D \succ_{AND\text{-}join} C \}$ and $RQ'(u_j) = \{ A \succ_{AND\text{-}join} C \}$. Applying Algorithm 1 to $RQ'(u_i) \cup RQ'(u_j)$ obtains the directed graph shown in Figure 8B. Although the directed graphs in Figure 2D and Figure 8B are different, they are semantically equivalent because the AND-join construct in Figure 8B limits activity C to start after both A and D have finished, and D should be started after A has finished. Users u_i and u_j do not encounter the problem of concurrent revision because they do not have to deal with edges and nodes while they are revising the workflow definition. While it is possible that the revised requirements contain CLWfDL rules that conflict with each other, the translator will also detect this situation and report it to the users.

V. THE CLWFDL DOCUMENT AND API

In this section we define how to represent CLWfDL rules in XML documents and a CLWfDL API. Developers can implement collaborative workflow definition systems in CLWfDL API. Figure 9 shows the syntax of a CLWfDL document (we specify syntax definitions in the Backus-Naur Form[3] [17] in this paper), which consists of the following two sections:

- *The header section*. Since a CLWfDL document is also an XML document, it begins with an XML declaration that specifies the version of XML being used (e.g., <?xml version="1.0"?>). This section also contains required namespace declarations.
- *The CLWfDL section*. This section contains CLWfDL rules. Each rule is composed of elements. The <priority> element specifies the priority of the rule, and is an integer. The <ruleType> element specifies the type of rules, which can be either "join" or "split". The <antecedent> and <consequent> elements are used to specify the antecedent and consequent activities of the rule.

```
CLWfDL  →
   <CLWfDL id="id">
      rule {rule}
   </CLWfDL >

   rule  →
      <rule user="user">
         <priority> priority </priority>
         <ruleID> μ </ruleID>
         <ruleType> join|split </ruleType>
         <antecedent> activityID </antecedent>
         <consequent> activityID </consequent>
         [<ifElse> predicate </ifElse>]
      </rule>

id → string , user → string
priority → integer , μ → string
activityID → string , predicate → string
```

*string represents a character string.

Figure 9. Syntax of the CLWfDL document

[3] Conventionally, a nonterminal symbol in Backus-Naur form is delimited by < and >. To prevent confusion with XML elements, nonterminal symbols are underscored in this paper.

Figure 10 shows two CLWfDL documents that are definitions from two users as shown in Figure 8B. We also propose the CLWfDL API in order to make it easy to implement the translator shown in Figure 3. The CLWfDL API was developed in the JAVA programming language and dom4j library [18], and can construct a directed graph G = (V, E) from some CLWfDL rules according to Algorithm 1 and Algorithm 2. We can then generate a workflow definition based on G for a specific workflow engine. The core classes are theCLWfDL, theTools, checkConflict, edge, node, and rule. A rule object corresponds to a single CLWfDL rule. The theCLWfDL and checkConflict classes implement Algorithm 1 and Algorithm 2, respectively.

```
<?xml version="1.0"
encoding="UTF-8"?>
<CLWfDL>
   <rule user=" u_i ">
      <priority>0</priority>
      <ruleID>AND-join</ruleID>
      <ruleType>join</ruleType>
      <antecedent>A</antecedent>
   <consequent>D</consequent>
   </rule>
   <rule user=" u_i ">
      <priority>0</priority>
      <ruleID>AND-join</ruleID>
      <ruleType>join</ruleType>
      <antecedent>D</antecedent>
   <consequent>C</consequent>
   </rule>
</CLWfDL>
```

(A)

```
<?xml version="1.0"
encoding="UTF-8"?>
<CLWfDL>
   <rule user=" u_j ">
      <priority>0</priority>
      <ruleID>AND-join</ruleID>
      <ruleType>join</ruleType>
      <antecedent>A</antecedent>
   <consequent>C</consequent>
   </rule>
</CLWfDL>
```

(B)

Figure 10. Two example CLWfDL documents

```
/* Invoke CLWfDL API to input CLWfDL documents and output
corresponding sets : Γ, Γ′, V and E. */

/* Step 1: create a CLWfDL object for process some CLWfDL
documents */
TheCLWfDL theCLWfDL = new TheCLWfDL();

// Step 2: read some CLWfDL document files from some users.
theCLWfDL.readXML("U_John.xml");
theCLWfDL.readXML("U_Alice.xml");
…

// Step 3: Create the Γ set defined in section II.
theCLWfDL.createGamma();

/* Step 4: Create the Γ′ set by check the CLWfDL definition
conflicts and try to eliminate them in Γ according to
Algorithm 2. */
theCLWfDL.removeConflict();

/* Step 5: We check if there are still rules in Γ′ which
conflict. */
if (theCLWfDL.checkConflict()) {
  RuleVector rV=theCLWfDL.getConflictedRules();
  For (rv)
{ rv.show();}
}

/* Step 6: Create the V set and E set of the CLWfDL directed
graph according to Algorithm 1. */
theCLWfDL.createDG();

// Obtain the created V set.
NodeVector nV=theCLWfDL.getNodeList();

// Obtain the created E set.
edgeVector eV=theCLWfDL.getEdgeList();
```

Figure 11. Codes invoking the CLWfDL API

Figure 11 demonstrates how to use the CLWfDL API to process a set of CLWfDL documents. When the main program starts, it first instantiates a CLWfDL object in step 1. In step 2, this object inputs some CLWfDL documents, and step 3 invokes the createGamma() method to construct a set Γ of CLWfDL rules from the input documents. Before creating the directed graph it is necessary to invoke removeConflict() in step 4 which is actually Algorithm 2. This creates the Γ′ set. If there are still conflicting rules, they can be checked using the checkConflict() method in step 5. The processing of the CLWfDL documents should be stopped if some rules are conflicting, and the corresponding users should be informed. Finally, step 6 invokes the createDG() method to create a set E of edges and a set V of nodes of a directed graph G according to Algorithm 1. The node and edge sets of the generated directed graph can be obtained by getNodeList() and getEdgeList(), respectively. Figure 12 shows the node and edge sets obtained by processing the CLWfDL documents in Figure 10. Directed graph G can be used to translate the workflow definition into another definition language. Table 3 shows the running time required for the CLWfDL API to process some documents with more rules. The CLWfDL API can be downloaded from http://www.csie.ntnu.edu.tw/~ghhwang/CLWfDL/CLWfDL_API_1_5.jar.

Node list	Edge list
[D]: activity node	[A] -> [D]
[A]: activity node	[AND_join_C] -> [C]
[C]: activity node	[D] -> [AND_join_C]
[AND_join_C]: AND-join node	[A] -> [AND_join_C]

Figure 12. Node and edge sets obtained by processing the CLWfDL documents in Figure 10

TABLE 3. RUNNING TIMES FOR PROCESSING SOME CLWFDL DOCUMENTS

Experimental Results	
11 rules	529 ms
12 rules with conflict	631 ms
32 rules with conflict	901 ms

VI. RELATED WORK

There are numerous popular workflow definition languages, including FDL [19], jPDL [20], XPDL [2], and BPEL [3], and they have corresponding graphical editors. A workflow editor presents a defined workflow as a directed graph; such editors include the IBM FlowMark workflow manager [19], JBoss jBPM [20], Enhydra JaWE [5], and BPEL Designer project [7]. Aalst and ter Hofstede proposed the YAWL [4] workflow language, and demonstrated that it can be used to define all the flow-control constructs existing in "workflow patterns" [10]. The YAWL system is still under development, but it already provides a graphical editor [21]. Since these workflow definition languages all employ a directed-graph-based editor to edit workflow definitions, we can consider them as directed-graph-based workflow definition languages. As mentioned in section I, a graphical workflow editor suffers from the concurrent-revision problem and does not provide an effective mechanism to implementing workflow definition by cloud collaboration.

The most famous application of cloud collaboration is Google Docs [8], which allows multiple users to edit a shared document in the cloud concurrently and distributively. Many researchers have recently proposed approaches for various applications that involve cloud collaboration. Vegesna proposed various concepts related to online collaboration and a specific online application called Zoho Notebook [22] that aims to effectively illustrate the advantages of online collaboration. It supports fine-grain access control and locking at the individual object level rather than the page level in collaborative document editing. Mikkonen and Nieminen discussed three major topics related to implementing a revision control system in the cloud collaborative development environment [23]. Their prototype system, called "Cored", allows users to edit codes concurrently and also communicate using means that are familiar from social media, which facilitates its development. This differs from a traditional version-control system because it reconsiders the role of version management based on the assumption that the background system can determine when a complete, runnable system exists by relying on compilation, testing, and integration capabilities available in the cloud. However, to the best of our knowledge, the present paper is the first to address the issue of defining a workflow process by cloud collaboration.

Stephenson et al. used a scenario to demonstrate an integrated environment for the development of business processes in the cloud [24]. They provided a collaborative model editor to model the activities of all the available experts. They proposed four basic experts—responsible for business, service, security, and the target platform—to develop and execute a business process complementarily. Each expert can create a BPMN model and share it with other experts. However, their model still has limitations. For example, if the business process is updated by one of the experts, this will impact on the other experts; this problem is very similar to the concurrent-revision problem that we describe in section I. In order to solve this problem, those authors suggested having a single person perform more than one role.

Jørgensen proposed a design guide for an interactive WfMS and described the challenges for interactive workflow modeling [25]. His goal was to use an interaction framework to help users to reinterpret the use of explicit process representations and the roles of models and components in WfMS architectures. That framework is based on the WORKWARE model language, which was inspired by the Action Port Model (APM), but it was simplified and had interactive enactment semantics added to it [26][27]. The basic assumptions are that the model only reflects parts of a socially constructed reality, and that it can be changed at any time. The users can interact with the system to redefine the workflow process. The APM uses a visualization editor to construct the model. The decision connectors (i.e., control nodes) of the APM cover AND/OR fork/joins. However, the framework cannot support concurrent revision from multiple users and thus is inappropriate for cloud-based cooperation.

A non-directed-graph-based workflow definition language has also been proposed. Glance et al. used generalized process structure grammars (GPSG) to generate flexible representations of collaborative processes and feature constraints to represent the artifacts of complex process [28][29]. They proposed writing constraints in GPSG rules to define the causal dependency between activities. Examples of the constraints mentioned in their paper include "precede", "<", "<=", ">", and ">=". They claimed that GPSG are more flexible than a traditional workflow definition language, since new rules can be added without needing to change the original rules, and rules can be revised independently of other rules. Our CLWfDL also has these advantages, and it seems that GPSG could be used in a cloud collaboration environment since the GPSG rules could be collected from different users to form the definition of a workflow. However, GPSG do not provide the ability to detect and solve conflicts between rules. Also, Glance et al. did not show whether GPSG rules can implement popular flow-control constructs and if it is possible to translate GPSG rules into directed graphs.

VII. CONCLUSION

Applying the working model of cloud collaboration to define a workflow in the CLWfDL can reduce the cost, since collaborating users do not have to be in the same space and human intervention is not needed to analyze the requirements of users. We believe that this approach has considerable potential in today's globalized society. The CLWfDL supports distributed definition and concurrent revision. Despite having only two types of rules, this new language can implement almost all flow-control constructs, as shown in [14]. After users learn the meaning of the two types of rules, they can use them to specify their requirements for workflow execution. A translator can detect if there is any conflict between the rules from multiple users. If conflicts exist, the involved users can be informed and then they can communicate and negotiate amongst themselves so as to revise their rules until there is no conflict. Each user maintains and is responsible for his/her own rules. We have proposed algorithms that can translate rules from multiple users into a directed graph, which can subsequently be transformed into any workflow definition language. This means that existing workflow engines can provide the workflow enactment service for workflows defined by CLWfDL rules. Since each user only specifies rules, we can design a home page for him/her to edit his/her own rules, and the system can present the final workflow definition in the form of a directed graph.

REFERENCES

[1] D. Georgakopoulos, M. Hornick and A. Sheth, "An overview of workflow management: from process modeling to workflow automation infrastructure," Distributed and Parallel Databases, vol. 3, issue 2, pp. 119–153, April 1995.

[2] WFMC, "Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface – XML Process Definition Language (XPDL) (WFMCTC- 1025)," Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 2002.

[3] OASIS, "Web Services Business Process Execution Language (WSBPEL)," http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, OASIS Standard, April 2007.

[4] W.M.P. van der Aalst, and A.H.M. ter Hofstede, "YAWL: yet another workflow language," Information Systems, vol. 30, issue 4, pp. 245–275, 2005.

[5] "JaWE - Java Workflow Editor," http://www.together.at/prod/workflow/twe.

[6] OMG "Business Process Model And Notation (BPMN)," http://www.omg.org/spec/BPMN/2.0/, Object Management Group, January 2011.

[7] "Eclipse BPEL Designer Project," http://www.eclipse.org/bpel/.

[8] "Google Docs," https://docs.google.com/.

[9] "Google Calendar," https://calendar.google.com/.

[10] "Workflow Patterns Home Page," http://www.workflowpatterns.com.

[11] "Google Forms," https://docs.google.com/forms/.

[12] P. M. Sant, "Exclusive read, exclusive write," in Dictionary of Algorithms and Data Structures, Paul E. Black, ed., U.S. National Institute of Standards and Technology. December 17, 2004.

[13] F. Casati, S. Ceri, S. Paraboschi, and G. Goodman, "Concurrency Control and Recovery in Database Systems," Addison Wesley Publishing Company, 1987, ISBN 0-201-10715-5.

[14] Chi Wu-Lee and Gwan-Hwan Hwang, "Workflow Definition by Cloud Collaboration," Chi Wu-Lee and Gwan-Hwan Hwang, Technical Report, National Taiwan Normal University, 2013. http://www.csie.ntnu.edu.tw/~ghhwang/TR/CLWfDL_Technical_Report_2013_08_06.pdf

[15] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and implementation of exceptions in workflow management systems," ACM Transactions on Database Systems, 24(3): 405-451, 1999.

[16] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler, "On Structured Workflow Modelling," The 12th International Conference on Advanced Information Systems Engineering (CAiSE), LNCS 1789, pp. 431-445, 2000.

[17] Control Data Corporation, "ALGOL-60 version 5 reference manual," CDC, 1979, Appendix D, Available at: <http://www.lrz.de/~bernhard/Algol-BNF.html>.

[18] "dom4j," http://dom4j.sourceforge.net/.

[19] "IBM FlowMark: Modeling Workflow," Version 2 Release 2. Publ. No. SH-19-8241-01, 1996.

[20] "JBoss, jBPM, jPDL," http://www.jboss.org/jbpm/.

[21] "YAWL System," http://www.yawlfoundation.org/.

[22] R. Vegesna, "Collaboration in Context: From the Desktop to the Cloud," The 2012 45th Hawaii International Conference on System Science (HICSS), pp. 669-673, January 2012.

[23] T. Mikkonen and A. Nieminen, "Elements for a cloud-based development environment: online collaboration, revision control, and continuous integration," The 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA), pp. 14-20, August 2012.

[24] B. Stephenson, J. Li, F. Lins, R. Medeiros, B. Silva, A. Souza, D. Aragao, J. Damasceno, P. Maciel and N. Rosa, "SSC4Cloud Tooling: An Integrated Environment for the Development of Business Processes with Security Requirements in the Cloud," The 7th IEEE 2011 World Congress on Services, pp. 53-60, July 2011.

[25] H. D. Jørgensen, "Interaction as a Framework for Flexible Workflow Modelling," Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work 2001, pp. 32-41, October 2001.

[26] S. Carlsen, "Action Port Model: A Mixed Paradigm Conceptual Workflow Modeling Language," CoopIS '98, New York, 1998.

[27] H. D. Jørgensen and S. Carlsen, "Emergent Workflow: Integrated Planning and Performance of Process Instances," Workflow Management '99, Münster, Germany, 1999.

[28] N. S. Glance, D. S. Pagani, and R. Pareschi, "Generalized process structure grammars GPSG for flexible representations of work," Proceedings of the 1996 ACM conference on Computer supported cooperative work, pp. 180-189, November 1996.

[29] N. Glance, Pagani, D., and Pareschi, R. "Generalized Process Structure Grammars for Modeling Collaborative Writing," Rank Xerox Research Centre, Grenoble, Technical Report March 1996.