Confucius: A Tool Supporting Collaborative Scientific Workflow Composition

Jia Zhang, Daniel Kuc, and Shiyong Lu

Abstract—Modern scientific data management and analysis usually rely on multiple scientists with diverse expertise. In recent years, such a collaborative effort is often structured and automated by a dataflow-oriented process called scientific workflow. However, such workflows may have to be designed and revised among multiple scientists over a long time period. Existing tools are single user-oriented and do not support workflow development in a "collaborative fashion." In this paper, we report our research on the enabling techniques in the aspects of collaboration provenance management and reproduciability. Based on a scientific collaboration ontology, we propose a service-oriented collaboration model supported by a set of composable collaboration primitives and patterns. The collaboration protocols are then applied to support effective concurrency control in the process of collaborative workflow composition. We also report the design and development of Confucius, a service-oriented collaborative scientific workflow composition tool that extends an open-source, single-user environment.

Index Terms— H.4.1.g Workflow management, M.4 Service-Oriented Architecture, H.5.3.c Computer-supported cooperative work

1 Introduction

The advancement of modern science has created sheer volume of data with increasing complexity. Processing and managing such large-scale scientific data sets is usually beyond the realms of individual scientists to solve [1]; instead, it has to rely on multiple domain scientists with diverse expertise. For example, the Large Synoptic Survey Telescope (LSST) experiment [2], which aims to repeatedly image half of the sky over a planned 10-year survey, produces data at a rate of 300 MB/s and will result in catalogs of about 130 TB of roughly 3×10^9 sources times 10 years worth of data. Analyzing such data sets demands a collaboration of a number of organizations with over 1,800 scientists and engineers engaged.

Such scientific data analysis and processing is usually structured and automated by a dataflow-oriented process called *scientific workflow*. In contrast to business processes that are control-flow oriented and orchestrate a collection of well-defined business tasks to achieve a business goal, scientific workflows are often dataflow-oriented and streamline a collection of scientific tasks to enable and accelerate scientific discovery [3, 4]. Researchers use scientific workflows to integrate and structure local and remote heterogeneous computational and data resources to perform *in silico* experiments [1, 5-7]. The increasingly important role of scientific workflows in modern science was emphasized in an article titled "Beyond the Data Deluge" published in *Science* [8]: "the rapidity with which any given discipline advances is likely to depend on how well the community acquires the necessary expertise in database, *workflow management*, visualization, and cloud computing technologies."

In short, scientific workflow and scientific collaboration are two key techniques to support scientific data analysis and management. The convergence of the two trends naturally leads to a concept that we coined as *collaborative scientific workflow* [9], meaning that multiple scientists are involved and collaborate in the entire lifecycle of a workflow: its design, revision, execution, monitoring, and management. As the first step, we focus on design-time collaboration, aiming to build techniques and a tool to support collaborative scientific workflow design, both synchronously and asynchronously.

Collaborative workflow design is usually critical for the success of a comprehensive workflow composition. A very simplified LSST experiment [2] represents a three-step workflow: data retrieval, pre-processing, and data modeling. Designing each step requires different expertise. Meanwhile, the serial relationship among the steps implies that the accuracy of the entire process is determined by the design of each step and the error propagation between them. While involving scientists with different capabilities focus on finding a local optimal design of a particular step, collaboration between them will find a global optimal design for the entire workflow.

Existing scientific workflow tools do not particularly support collaborative composition. In our earlier article [9], we studied a number of representative scientific workflow management systems (SWFMSs) [10-15], including Kepler [10], Taverna [11], Triana [12], VisTrails [13], Pegasus [4], Swift [14], and VIEW [15, 16]. Our investigation found that they are all single user oriented, focusing on helping individual scientists construct workflows from available applications and services. Facilities that support scientists in collaboratively designing workflows are limited. Individual work artifacts (scientific workflows) are

- *Jia Zhang is with the Department of Computer Science, Northern Illinois University, DeKalb, IL 60115. E-mail: jiazhang@cs.niu.edu.*
- *Daniel Kuc is with the Department of Computer Science, Northern Illinois University, DeKalb, IL 60115.*
- *Shiyong Lu is with the Department of Computer Science, Wayne State University, Detroit, MI. E-mail: shiyong@wayne.edu.*

Manuscript originally received on 11/6/2010; revision received on 9/16/2011.

manually sent to collaborators (e.g., via emails) or uploaded to some shared social space (e.g., MyExperiment [17]) to enable collaborative design and discussion. For example, a collaborator can download a published workflow (e.g., in the description language provided by Taverna [11], a popular scientific workflow tool) from MyExperiment, load it into a local Taverna workbench, update it, and upload the revised workflow back to MyExperiment. Afterwards, other collaborators can continue to perform further changes and thus realize a collaborative design.

Such a discrete collaboration style obviously does not support real-time discussion and collaboration, which are usually critical to scientific exploration. In addition, provenance data (workflow design history) is not maintained sufficiently, which is critical for workflow reproducibility. Therefore, we have been developing a design environment, equipped with system-level support for collaborative scientific workflow composition. To memorize a famous ancient Chinese philosopher and educator, we name our system after his name as Confucius. Without reinventing the wheel, we examined a widely used single-user scientific workflow tool, Taverna [11], and extended it into a multi-user version.

To our best knowledge, this is the first effort of successfully realizing a collaborative scientific workflow composition workbench. In this paper, we focus on reporting our design and development of one key enabling technique, collaboration protocol. We propose a scientific collaboration provenance ontology, and based on it, we have developed a service-oriented collaboration model that is supported by a set of composable collaboration primitives and patterns. The collaboration protocols are then applied to support effective concurrency control in the process of workflow co-design.

The core idea of Service Oriented Architecture (SOA) is to position services as the primary means (components) that encapsulate solution logic [18] as reusable assets. In this project, we have leveraged the concept of SOA to build our system for higher interoperability, reusability, and productivity. Since we focus on scientific workflows, throughout this paper, we use the terms *scientific workflows* and *workflows* interchangeably.

The remainder of the paper is organized as follows. In Section 2, we present related work. In Sections 3, 4, and 5, we introduce collaboration models, collaboration protocols, and composition concurrency control mechanisms. In Section 6, we discuss system implementation and experimental study. In Section 7, we draw conclusions.

2 Related Work

In this section, we compare our approach with related work in three categories: scientific workflow management systems, business workflow coordination, and collaborative workflow composition. Note that we focus on the workflow design phase.

2.1 Scientific Workflow Management Systems

To date, several scientific workflow management systems (SWFMSs) have been developed as single-user environments that help individual scientists construct workflows from available scientific resources. Representative SWFMSs include Kepler [10], Taverna [11], Triana [12], VisTrails [13], Pegasus [4], Swift [14], Trident [19], and VIEW [15, 16]. Each system shows unique features.

Kepler [10] models a workflow as a composition of components called *actors*, controlled by a *director*. Taverna [11] uses an XML-based workflow language called *SCUFL/XSCUFL* for workflow representation, with each component being either a Web service or a processor developed using Java Beanshell script. Triana [12] provides a sophisticated graphical user interface for workflow composition and modification, including grouping, editing, and zooming functions. VisTrails [13] focuses on workflow visualization supporting provenance tracking of workflow evolution and data product derivation. Pegasus [4] provides a framework that leverages artificial intelligence planning techniques to map complex scientific workflows onto distributed Grid resources. Swift [14] uses a scripting language called *SwiftScript* to support specification of large-scale computations over a Grid. Trident is a scientific workflow workbench built on top of a commercial workflow system, Windows Workflow Foundation (WF) included in the Windows operating system. VIEW [15, 16] system features efficient provenance management based on RDFProv [20, 21] that combines advantages of Semantic Web technologies with relational databases [21, 22].

Each of the SWFMSs provides a platform to support individual scientists in composing. Some systems show some collaboration features, in the sense that they allow a scientist to compose a workflow from shared resources and services. However, they provide limited support for multiple scientists to collaboratively compose a shared workflow [23]. For example, Taverna [11] users can publish their composed workflows in a dedicated social workflow space (e.g., MyExperiment [17]); others can download the workflows and load them using the same SWFMS, make changes, and upload the new versions into MyExperiment to initiate further interactions. However, such SWFMSs do not support real-time workflow co-design.

In contrast to the existing SWFMSs, we study how to establish a system that enables multiple scientists to collaboratively compose workflows with system-level support.

2.2 Business Process Coordination

For business workflows, the term "collaborative workflows" is interchangeable with the term "coordinated workflows" [24-26]. They emphasize coordination between workflows. In contrast, we use the term to emphasize the involvement of multiple scientists with a workflow.

A number of work has been conducted to help model and compose business workflows [27]. Huang et al. [28] and Dang et al. [29] employ agents technology to reason about the composition of coordinated workflows. Balasooriya et al. [30] propose a decentralized services-oriented middleware architecture to compose workflows

and specify their dependencies. Balasooriya et al. [31] propose a two-layer framework, where Web services are modeled as self-coordinating entities, and a workflow interconnects such entities into a network of objects. Business Process Models (BPM) [32] enables Web form-style asynchronous collaborative business process modeling.

Some researchers particularly study modeling of workflow coordination in a Grid environment. Miller et al. [33] propose a language to specify workflow composition, comprising both reactive tasks (Web services) and proactive tasks (autonomous agents) in a Grid environment. Based on γ-calculus, Nemeth et al. [34] model workflow coordination as molecules and reactions to enable autonomous evolution in a changing Grid environment.

In contrast to business process modeling where comprising components are directly connected through control links, tasks in scientific workflow modeling are connected through data links, which have to be considered when locks are assigned to tasks during collaborative workflow design. We thus introduce a concept of synchronization area (defined in Section 5.1).

Lu and Sadiq [27] consider pre- or post-conditions for each comprising task, and use event-condition-action (ECA) rules to specify some runtime behavior at design time. Compared to their approach, we define human collaboration relationships at design time.

The Computer Supported Cooperative Work (CSCW) community has studied the general-purpose concurrent design problem, where collaborators with different ownerships possess different controls over the shared work product [35, 36]. To name a few, OntoEdit [37] supports a collaborative software engineering process; Yen et al. present a collaborative design tool that allows privileged collaborators to change the process [35]; OPCATeam [36] integrates the object-oriented and process-oriented paradigms into one single framework to enable the coexistence of structured processes and human interaction behaviors in one business process modeling system. In contrast, our work focuses on dataflow-oriented collaboration, where semantic relationships and constraints between different comprising components (e.g., tasks and data links) need to be carefully considered during concurrent composition.

2.3 Collaborative Workflow Composition

The business community recently recognized the need of involving humans into business workflows and has developed a preliminary model [38]. Ayachitula et al. [39] divide workflows into human-centric workflows and automated process-based workflows. Russell et al. [40] propose to establish a separate team access control layer, which combines role and organization, to manage access in a collaborative workflow environment. The BPEL4People [38] workflow model is proposed to extend the *de facto* industry standard business workflow language BPEL [41] to standardize the interaction between automated and human workflows.

However, these business workflow-oriented models are not suitable to be used for supporting collaborative scientific workflows because, business workflows are controlflow-oriented and hence lack dataflow constructs for interaction, movement, and processing of large datasets. Furthermore, provenance data management for the reproducibility of scientific results is essential for scientific workflows but not for business workflows. Hence, scientific workflows pose a different set of requirements [5]. For BPEL4People specifically, every computational component in BPEL must be a Web service, thus, it lacks the support of modeling user interaction and visualization intensive tasks.

Sayah and Zhang [42] present their annotated business hyperchain technology that enables on-demand business collaboration with the Web services technology. They propose a set of business collaboration primitives to serve business scenarios. In contrast, our collaboration primitives serve scientific collaboration scenarios. In addition, our work emphasizes design-time collaboration provenance capturing for credit acknowledgement as well as guiding future collaborative workflow composition.

We surveyed the state of the art of the field of scientific workflows toward the support of collaborative scientific workflows [9]. Our observations directly motivated the research work reported in this paper. We also have surveyed the literature of workflow control mechanisms in a collaborative environment in [43]. Our study helped us build a linkage between scientific workflow and collaborative work.

Our "collaboration provenance" is different from the term "collaborative provenance" used by Dr. Altintas et al. [44]. Their "collaborative provenance" means, "inferring dependencies across multiple workflow runs and understanding user collaborations based on scientific workflow runs"[44]. In contrast, our term "collaboration provenance" aims to capture how scientists collaboratively design a common scientific workflow, e.g., who has designed which part.

3 Collaborative Scientific Workflow Composition Model

Provenance has been widely considered critical to the reproducibility of scientific workflows [45, 46]. Compared to existing significant amount of work focusing on provenance for run-time workflow execution, our work focuses on collaboration provenance that tracks human interactions and efforts in the process of scientific workflow composition. Our method is to record all collaborative activities that contribute to a composed workflow.

3.1 Provenance Ontology

We have developed a provenance ontology to support the modeling of various provenance data recording scientific workflow design and user interactions during the process of a collaborative workflow design. As shown in Fig. 1, our ontology is centered upon the concept of "workflow." Each scientific workflow comprises organized processors (tasks) and data links (aka. data channels), sub-workflows, as well as predefined requirements and annotations (comments). Each workflow maintains one or more floors that are tokens to ensure concurrency control. Long-term

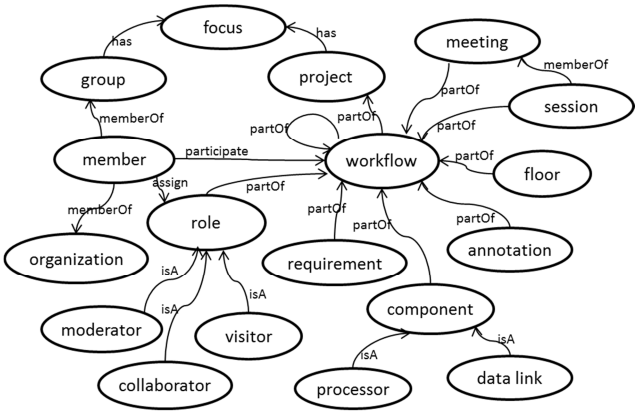

Fig. 1. Collaborative workflow composition provenance ontology.

collaboration on a scientific workflow forms a *meeting*. A short-term synchronous collaboration is called a *session*.

Each scientific workflow belongs to a project. Each project belongs to a scientific group (could be a virtual group). Each group may comprise multiple focuses, each involving multiple projects. A group contains a set of members, each may belong to different organizations.

Collaborative composition on a scientific workflow is conducted by members serving in different roles. The initiator (creator) of a scientific workflow is called a *moderator*. Scientists who cooperate on the lifecycle of a workflow are called *collaborators*. They have read and write privileges. A collaboration may also involve *visitors*, who are granted with read privilege only.

Our ontology, which is extensible, serves as a foundation for managing collaborative workflow design provenance. Each scientific collaboration project may define customized ontology and add additional concepts into the basic ontology for special purposes. For example, a research project may introduce project-wise particular roles in their collaboration.

3.2 Collaborative Composition Model

We designed models to regulate how collaborators can collaboratively design and update mutual workflows. Instead of reinventing the wheel, we chose to explore how to extend the single user-oriented Taverna tool.

3.2.1 Basic collaboration model

We carefully studied the latest Taverna code (version 2.0), focusing on exploring the feasibility of extending it into a collaborative version. As a starting point, we examined the communication paths between Taverna instances. In other word, we aim to find a way to allow two Taverna running instances to communicate with each other. We found that Taverna is built on top of an event-based mechanism, meaning that any user event (e.g., clicking a button) triggers a backend action. When a user chooses to save a workflow, Taverna will serialize the workflow as a file in XScufl [47] that is an XML-based workflow specification language. Fig. 2 shows a segment of XScufl code that contains one workflow with one to many dataflows, each comprising a sequence of elements including: name, a set of input ports, a set of output ports, a list of processors (tasks), some conditions, a set of data links (edges), and annotations. When a user selects to open such an XML file, the stored workflow will be loaded into the Taverna workbench and rendered on the screen.

```
<schema>
  <complexType name="Workflow">
    <sequence>
      <element name="dataflow" type="tav:Dataflow"
maxOccurs="unbounded" minOccurs="1"/>
...
  </complexType>

  <complexType name="Dataflow">
    <sequence>
      <element name="name" type="string"/>
      <element name="inputPorts"
type="tav:AnnotatedGranularDepthPorts"/>
      <element name="outputPorts" type="tav:Ports"/>
      <element name="processors" type="tav:Processors"/>
      <element name="conditions" type="tav:Conditions"/>
      <element name="datalinks" type="tav:Datalinks"/>
      <element name="annotations" type="tav:Annotations"
maxOccurs="1" minOccurs="0"/>
    </sequence>
  </complexType>
</schema>
```

Fig. 2. A segment of XScufl code.

Therefore, we utilized a file system-based workflow storage mechanism to enable communication between two Taverna versions. When a collaborator with write privilege saves a workflow, its serialized XML document can be propagated to another site where another collaborator has read privilege. An automatic file open action will render the same workflow on the reader's screen.

We adopted the observer design pattern [48] to build a preliminary infrastructure to enable collaboration between multiple Taverna versions. The observer pattern is a subset of the asynchronous publish/subscribe design pattern. A special subject is used to maintain a list of its dependents (observers) and automatically notify them of any state changes. Fig. 3 shows such a client/server-based infrastructure. A central server is established as the subject and maintains all collaborators' information, and all collaborators act as observers. As shown in Fig. 3, the central server also stores and manages all provenance data, so that late comers can view shared workflows.

Fig. 3 shows several possible flow scenarios. Client 1 registers a collaboration Group 1 on the central server (Step 1). Upon approval (Step 2), Client 1 shares a port to his/her potential collaborators (Step 3). Users from the invitation list (e.g., Client 2) may subscribe (Step 4) to the registered collaboration group (i.e., Group 1) and start to update the shared workflows within the group (Step 5). Any updated version will be stored in the central server and automatically distributed to all collaborators in the collaboration group (Step 6).

Any action in the original Taverna workbench (i.e., adding/deleting/updating an element, and saving a workflow) will trigger an automatic "save" action. The server will deliver the up-to-date workflow file to all participating collaborators.

3.2.2 Advanced collaboration model

Scientific collaborations usually last for a long period of time, e.g., months and years. In addition, temporary discussion groups and sessions may be formed in the lifecy-

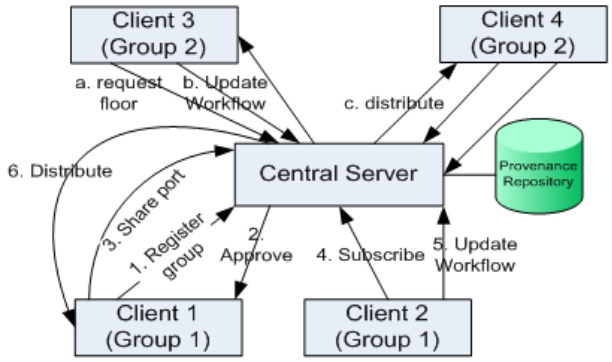

Fig. 3. Collaborative composition model.

cle of a long-term scientific collaboration process. Therefore, we constructed a hierarchical structure for the central server. It may host multiple collaboration groups, which may or may not have nesting relationships between them. The central server maintains all collaboration group information and acts as the subject for all registered groups. All observers (collaborators) are organized into corresponding collaboration groups. The central server also stores and manages all provenance data, so that it becomes a repository of workflow products and enables scalability. In other words, we realize a multi-tenancy infrastructure.

Within a collaboration group, a straightforward way is to allow everyone to do anything on a workflow at any time, and distribute the results to everyone in the same group. In the real life, however, typically only one person is allowed to speak at a certain moment in a group [49]. Thus, we grant access control policies so that only one person at a time can modify the shared workflow products and distribute the changes in the group.

We adopted the floor control technique from an extensively tested and well proved human communication protocol, Robert's Rules of Order (RRO) [49], where a single floor is maintained in a shared meeting environment. Each member requests and competes for the floor, and only the person who obtains the floor can talk in the meeting. Applying RRO to our collaboration environment, each member in a collaboration group must request a floor to gain the write privilege of the shared workflow products in the group. Otherwise, the changes will be kept locally and will not be distributed to other collaborators. A simple role-based model is adopted. The person who registers a collaboration group at the central server becomes the moderator of the group, and will automatically have the control over the floor. In this section, we discuss single-floor protocol. Multiple floor-based access control facility, with finer-grained locking mechanisms for higher concurrency, will be discussed in Section 5. As shown in Fig. 3, Client 3 in Group 2 requests the floor (Step a). Upon approval, Client 3 may update the workflow (Step b) and the changes will be distributed to other collaborators in the same group (e.g., Client 4) instantaneously.

The pseudo code, presented in algorithm 1, realizes a floor granting process. If the floor is not occupied, the requestor will be granted the floor exclusively; otherwise, the requestor will be put into the corresponding waiting list and wait for the floor. Upon releasing a floor, the requestor at the top of the waiting list will be automatically informed and granted the floor. If there is no one in the waiting list, then nothing will happen. The pseudo code (algorithm 2) shows a floor releasing process. A moderator may deprive the floor from a collaborator under certain circumstances, for example, if the collaborator loses her Internet connection.

Algorithm 1: Floor Granting Algorithm
Input: A collaborator releases a floor
Requirements: Release a floor.

```
1: check(waiting_list)
2: if (waiting_list ≠ Ø) then
3:    requestor ← get_top_requestor(waiting_list)
4:    floor_owner ← requestor
5:    notify(members)
6:    remove(requestor, waiting_list)
7: else if (waiting_list = Ø) then
8:    floor flag ← unoccupied
9:    notify(members)
10:endif
```

Algorithm 2: Floor Releasing Algorithm
Input: A requestor requests a floor
Requirements: Decide whether a floor should be granted.

```
1: check( floor)
2: if (floor ≠ taken), then
3:    floor flag ← occupied
4:    floor owner ← requestor
5:    notify(members)
6:    return true
7: else if (floor = taken) then
8:    insert(requestor, waiting_list)
9:    return false
10: endif
```

3.2.3 Light-weight collaboration model

The aforementioned client/server model represents a formal collaboration mode, as all communications are stored in a centralized server with permanent provenance storage. In contrast, sometimes researchers may prefer a more informal collaboration mode. Backdoor communications may occur among some team members in a free and private manner. In addition to free text conversations that can be supported by applications like instant messengers (IMs, which we have integrated into Taverna), here we focus on discussing how to share temporary workflow changes among a subset of collaborators.

To realize the backdoor collaboration, a straightforward way is to adopt the traditional peer-to-peer (P2P) mode, where each peer (i.e., Taverna instance) is equally weighted and is enabled to communicate with each other. This implies that each SWFMS instance becomes heavy-weight, meaning that we have to physically embed P2P communication code into each SWFMS instance. Recall that our centralized client/server model intentionally keeps each SWFMS client light weighted. This heavy-weight SWFMS client requirement will constrain the reusability and flexibility of the code of the SWFMS instance. In addition, our server-based communication mode is not reusable in this option.

To overcome these limitations, we propose a light-weight server model. A light-weight server is established

to temporarily store the latest version of the workflow product and broadcast it to all participating clients (i.e., SWFMS instances). We modularize the light server as a pluggable component to the original SWFMS instance code. This implies that such a server can run on every client side, in addition to the light-weight client. Between two peers who intend to communicate, only one peer has to initiate a light-weight server.

Fig. 4 illustrates the light-weight communication protocol between two collaborators using a UML sequence diagram. When one user (Collaborator1) wants to start a back-door channel, she sends an invitation (to Collaborator2). Upon receiving an agreement, the user (Collaborator1) implicitly instantiates and starts a light-weight server at the user side. A signal is sent to the other party as well. It is in a light-weight mode, in the sense that it does not permanently store workflow products. As shown in Fig. 4, when Collaborator1 makes some changes to the shared workflow, the changes will be submitted to the light-weight server. The light-weight server will in turn propagate the changes to the participating Collaborator2. Similarly, when Collaborator2 makes changes, they will be propagated to Collaborator1 through the light-weight server. Finally, when the initiator (Collaborator1) decides to finish the backdoor communication, the final version of the changes can be sent to the central server to store, if so desired.

4 Collaboration Protocols

Business process modeling techniques [27] use rules to specify runtime behavior of workflows at design time. Similarly, we model run-time collaboration specifications (e.g., rules, patterns, and primitives) at design time. We focus on how to enable recording such collaboration design and their changes.

4.1 Collaboration Rules

The granularity of a collaboration happens at either dataset or task level. Different research projects may adopt different collaboration rules; thus, a collaboration model must be configurable. We propose a 4-tuple collaboration rule container as shown in Fig. 5:

$$C-Rule =< Owner, Operator, Monitor, Validator >$$

The collaboration container comprises four basic plug-in roles: owner, operator, monitor, and validator. Plug-in roles mean that they represent role types, and zero to multiple role instances may be created at run-time. An

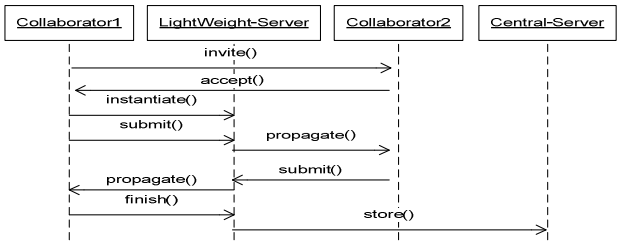

Fig. 4. Light-weight collaborative composition model.

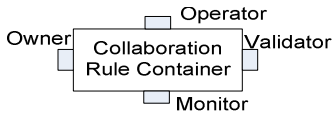

Fig. 5. Collaboration rule container.

owner role represents a group of scientists who have ownerships over a dataset or a task. An operator role represents a group of scientists who have the privilege to operate on a dataset or a task. A monitor role represents a group of scientists who have the privilege to monitor the operation process of a task or over a dataset. A validator role represents a group of scientists who have the privilege to validate an operation over a dataset or a task and claim the success/failure of such an operation. Note that what operators, monitors, and validators have to do is project-specific. One scientist may act in multiple roles simultaneously in a workflow composition process.

A collaboration rule specifies an instance of a rule container, with the participating collaborators at run time. Because of the exploratory nature of scientific workflow design, such a run-time collaboration rule is a design-time expectation, and will be recorded in the format of annotations attached to particular parts of workflows, as illustrated in Fig. 1.

4.2 Collaboration Patterns

Different from business processes, a scientific workflow is exploratory, thus requiring constant human interaction and intervention. For example, a task run may require validation before its subsequent tasks can continue. Such a requirement should be recorded as part of the workflow design.

We studied a number of known scientific collaboration projects documented in [1] to identify data-centric collaboration patterns. As a starting point, we summarized a set of six collaboration patterns: (1) dataset request, (2) analysis request, (3) validation request, (4) discussion request, (5) co-run, and (6) co-approve. We first introduced the six patterns for *two-way collaboration*, where two scientists are involved in a collaborative activity [50]. Our preliminary experiences show that these patterns satisfy basic collaboration requirements. Then, we extended the patterns to multi-way collaboration. In the near future, when our system is applied to the real-world collaborative projects, we plan to study and elicit more collaboration patterns.

The dataset request pattern reflects a scenario when some specific data is required, during the execution of a scientific experiment, while the dataset belongs to an external scientist group. Given a workflow W, scientist A asks for dataset D from scientist B before continuing.

The analysis request pattern reflects a scenario when some particular data obtained has to be analyzed by a specific tool or process that is owned by an external scientist group. Given a workflow W, scientist A asks scientist B to analyze dataset D.

The validation request pattern reflects a scenario when an interesting discovery is reached that requires verification and validation by a group of scientists with specific expertise. In the context of a workflow W, scientist A asks scientist B to validate a specific task T or dataset D. The

result will be either positive or negative.

The discussion request pattern reflects a scenario when discussion is needed over some specific topics, and the results of the discussion will decide the direction (or steps) of the following actions. In the context of a workflow *W*, scientist *A* identifies a group of scientists to discuss over a task *T* or a dataset *D*.

The co-run pattern reflects a scenario when scientists individually run proprietary data analysis processes over the same dataset simultaneously. Given one dataset *D*, scientists A, B, …, N perform workflows $W_1, W_2, \ldots W_n$ concurrently and respectively, and then compare the results obtained from their workflow runs.

The co-approve pattern reflects a scenario when scientists have to reach an agreement on an experimental result.

These collaboration patterns can be represented using our rule container to realize a fine-grained collaboration control. For example, the data analysis pattern can be represented by a rule with owner A and operator B.

4.3 Collaboration Primitives

TABLE I. COLLABORATION PRIMITIVES

Type	Primitive Name
Collaboration preparation primitives	Request for Dataset (RFD)
	Request for Data Analysis (RFA)
	Request for Validation (RFV)
	Request for Discussion (RFC)
	Request for Co-run (RFCR)
	Request for Co-approval (RFCA)
Collaboration conduction primitives	Accept or Reject Request (A/R)
	Command Submission (CS)
	Data Submission (DS)
	Update Submission (US)

Based on the collaboration patterns, we designed a set of semi-structured collaboration primitives, as summarized in Table I. The primitives are divided into two categories: collaboration preparation primitives and collaboration conduction primitives. Since scientific collaboration may last for a long period of time, we adopt an asynchronous communication mode, meaning that each collaboration primitive is associated with an instant acknowledgement.

Six collaboration preparation primitives are constructed: (1) Request for Dataset (RFD), when a dataset is needed during a workflow; (2) Request for Data Analysis (RFA), when a data analysis process is needed during a workflow; (3) Request for Validation (RFV), when a data validation process is required; (4) Request for Discussion (RFC), when a discussion is required; (5) Request for Co-run (RFCR), when concurrent sub-workflows are required; and (6) Request for Co-approval (RFCA), when an approval has to be made by multiple parties.

Four collaboration conduction primitives are identified: (1) Accept or Reject Request (A/R), when a request (e.g., RFD) is accepted or rejected by a collaborator; (2) Command Submission (CS), when a specific computational command is provided; (3) Data Submission (DS), when a specific data set is transferred; and (4) Update Submission (US), when a collaborator updates collaboration status in response to a request.

In addition to be used individually, these collaboration primitives can be used as building blocks for collaborators to model more comprehensive collaboration patterns.

4.4 Collaboration Mini-Workflow

Based on the established collaboration primitives, we apply the SOA concept to implement the collaboration patterns. Each collaboration pattern is accomplished by a mini-workflow comprising a set of configured collaboration primitives. Through different combinations of the set of collaboration primitives, different collaboration patterns can be realized.

We have constructed six example mini-workflows to realize the six collaboration patterns described in Section 4.2. (1) dataset request: comprising the RFD, A/R, and DS primitives; (2) analysis request: comprising the RFA, A/R, and CS primitives; (3) validation request: comprising the RFV, A/R, and CS primitives; (4) discussion request: comprising the RFC primitive and a collection of US primitives; (5) co-run: comprising the RFCR, A/R, and US primitives; and (6) co-approve: comprising the RFCA, A/R, and US primitives.

Such a mini-workflow can be formalized using the Business Process Execution Language (BPEL). Since BPEL is based on Pi-calculus, modeling mini-workflows in BPEL will allow us to formally reason about the

```
<process name"RFDmicroflow"
   targetNamespace="urn:CollaborationConstructs"
   xmlns:tns="urn:samples:CollaborationConstructs"
   xmlns="http://confucius.org/constructs/">

   <sequence>
    <invoke name="invokeRFD"
       partner="CollaboratorA" portType="tns:RFDoriginatorPT"
       operation="sendRFD" outputVariable="RFD">
    </invoke>

    <invoke name="ackRFD"
       partner="CollaboratorB" portType="tns:RFDreceiverPT"
       operation="ackRFD" outputVariable="RFD_Receipt_Ack">
     </invoke>

    <invoke name="acceptRFD"
       partner="CollaboratorB" portType="tns:RFDreceiverPT"
       operation="acceptRFD " outputVariable="A">
    </invoke>

    <invoke name="ackAcceptRFD"
       partner="CollaboratorA" portType="tns:RFDoriginatorPT"
       operation="ackAcceptRFD" outputVaria-
ble="A_Receipt_Ack">
    </invoke>

    <invoke name="invokeDS"
       partner="CollaboratorB" portType="tns:RFDreceiverPT"
       operation="submitDS" outputVariable="DS">
    </invoke>

    <invoke name="ackDS"
       partner="CollaboratorA" portType="tns:RFDoriginatorPT"
       operation="receiveDS" outputVariable="DS_Receipt_Ack">
    </invoke>
   </sequence>
</process>
```

Fig. 6. Service-oriented mini-workflow.

construction of a new collaboration pattern. Such a validation allows for design-time static checking.

Taking the first collaboration pattern (dataset request) as an example, Fig. 6 shows a section of its BPEL specification. For simplicity, we skipped the section defining messages (collaboration messages), partners (collaborators A and B), and variables (messages), and links (expressing synchronization dependencies). As shown in Fig. 6, each collaboration primitive is wrapped as a Web service. Two parties (Collaborators A and B) act as service providers and service requestors, respectively. Each collaboration primitive is realized by a service call, associated with the corresponding messages. Once represented by BPEL, multiple collaboration constructs may be combined to form a comprehensive collaboration scenario. Such a service-oriented model enables platform-neutral and language-neutral collaboration.

4.5 Service-Oriented Collaboration Provenance

We decided to adopt the Web services technology [18] to realize collaboration. As the best enabling technology of SOA to date, it allows us to design collaborations among participating scientists with platform independence and language independence. Collaboration primitives are encapsulated in Simple Object Access Protocol (SOAP) messages and communicated between collaborators. To enable validation and analysis, we adopted the XML Schema to uniform the format of collaboration messages. Fig. 7 shows a section of the specification of a collaboration message.

Messages are divided into request messages and response messages. Each message contains one or more primitives that form a transaction, meaning that they form an atomic unit of work in a scientific workflow. Each transaction aims to serve for a task in a workflow, which belongs to a scientific project. A message may also contain optional data such as comments.

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://confucius.org/class"
            xmlns:tns="http://confucius.org/class">
  <xsd:element name="Name" type="xsd:string"/>
  <xsd:element name="task" type="xsd:anyURI" de-
fault="http://confucius.org/class#Task"/>
  <xsd:element name="workflow" type="xsd:anyURI" de-
fault="http://confucius.org/class#Workflow"/>
  <xsd:element name="project" type="xsd:anyURI" de-
fault="http://confucius.org/class#Project"/>
  <xsd:element name="Construct">
    <xsd:annotation>
      <xsd:documentation> A construct is the atomic unit of
collaborative work in a scientific workflow.
</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="tns:Name"/>
        <xsd:element ref="tns:task"/>
        <xsd:element ref="tns:workflow"/>
        <xsd:element ref="tns:project"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Fig. 7. Schema for a transaction.

Such collaboration designs are recorded as provenance in the format of annotations (as shown in Fig. 1) attached to the workflow under construction.

5 Workflow Composition Concurrency Control

At composition time, multiple scientists collaborate to develop a scientific workflow. Thus, concurrency control deserves consideration to ensure design productivity.

5.1 Locking Granularity

Adopting the instrument from an extensively tested and well proved human communication protocol, RRO [49], we originally established a workflow-level floor control mechanism as described in Section 3.2.2. Each collaborator competes for the shared floor before making any changes.

Such a workflow-level floor control may not be efficient to support large-scale scientific workflow composition, though. Since scientific research is an exploratory process, the development of a workflow may undergo many rounds of discussions and changes and may last for a long period of time. Meanwhile, a collaboration group nowadays usually comprises scientists from different organizations at distributed locations. They may possess different schedules and may even reside in different time zones; thus, their collaboration may adopt both synchronous and asynchronous modes. Furthermore, a large-scale scientific workflow may involve many comprising components. It is neither efficient nor practical, if one scientist working on one component locks the entire workflow and prevents other scientists from working on unrelated components.

To increase composition concurrency, we investigated the option of locking the smallest building blocks. A scientific workflow allows multiple non-overlapped locks, so that multiple scientists may work on the corresponding locked components simultaneously.

According to the existing SWFMS tools, the smallest building blocks in a workflow are tasks and data channels. For example, in Taverna, a task is called a *processor*; a data channel linking between processors is called a data link. Fig. 8 is a highly simplified scientific workflow drawn in Taverna, which illustrates a word count example using the MapReduce programming model [51]. Two processors *Mappers* repeatedly process a list of word lines, by breaking each line into individual words and generating a list of <word, 1> pairs over all the words found. All the intermediate <word, 1> pairs are transferred, through the data links, to the processor *Reducer* that aggregates the pairs according to the words. The results are a list of <word, value> pairs that show the number of appearances of each word.

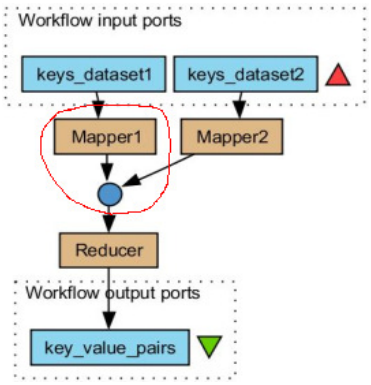

Fig. 8. Word count workflow.

If we set up the locks on individual processors and data links only, two collaborators may concurrently update one processor *Mapper1* and its output data links, respectively. This situation may not be desirable, because the data link directly depends on the processor. In other words, connected processors and data links may have close semantic relationships, which need to be preserved by requiring that adjacent entities cannot be updated by different collaborators at the same time.

Furthermore, continuous processors in a workflow may also possess semantic relationships between them. For example, as shown in Fig. 8 where triangles represent workflow inputs and outputs, the *Mapper1* processor and the *Reducer* processor are neighbors in the workflow, and a data link connects them together. The *Reducer* processor stays at the downstream of *Mapper1*; meaning that the output of the *Mapper1* processor serves as the input of the *Reducer* processor. Assume that two collaborators are working on the two processors simultaneously, and collaborator *A* changes some business logic at the *Mapper1* processor. Even if these changes may not change the input format of the *Reducer* processor, the collaborator working on the *Reducer* processor should be aware that someone is working on the upstream processor.

Therefore, we introduce a concept of *synchronization area* that represents a conceptual area in a shared scientific workflow, which allows only one collaborator to work on it at a given time. Such an area represents a semantic area. In the context of a Taverna workflow, if a user tries to lock a data link, the synchronization area is the data link itself. If a user tries to lock a processor, the synchronization area will be dynamically delimited and include all of the fan-out data links of the processor. In Fig. 8, the manually drawn red circle around the *Mapper1* processor and its fan-out data link represents such a synchronization area.

5.2 Locking Algorithms

Based on the concept of synchronization area, we developed four algorithms (3-6), on locking/releasing a processor and locking/releasing a data link.

If a user selects a processor to lock it, we first check whether it has been locked by another collaborator. If nobody locks it, then an uninterruptable transaction starts. First, we set the lock flag of the processor, and fill the name of the owner of the processor. For each outgoing data link of the processor, we check whether there is an active lock on it. If any outgoing data link is currently locked by other collaborators, the entire locking attempt will be aborted. Otherwise, we call the corresponding algorithm (i.e., 5) to lock the data link. After all outgoing data links are locked, the transaction succeeds. In summary, the lock action will automatically lock all downstream data links, in addition to the processor.

To release a processor, we will first check whether the user has the privilege, i.e., whether she is the owner of the processor. If the answer is positive, in addition to the processor itself, the action will call the corresponding algorithm (i.e., 6) to release all of its downstream data links.

To lock a data link, we first check whether the data link has been uploaded into the database (here we adopt a lazy instantiation pattern for a higher performance). After ensuring that the data link is in the locked link list, we check whether it has already been locked. If not, the data link will be marked as being locked. Otherwise, a notification will be sent. To release a data link, we first check whether the user has the privilege, i.e., whether she is the owner of the data link. If the answer is positive, the flag of the data link will be set as unlocked.

Algorithm 3: Lock Processor
Input: A user selects a processor and presses "lock"
Requirements: Lock a processor.
1: **if** processor ∈ locked processor list **then** do nothing;
2: **else**
3: **begin transaction**
4: processor_owner ← self; lock_flag ← 1
5: **for** each outgoing data link
6: **call** lock_data_link
7: **if** return = false **then abort**
8: **end transaction**
9: **endif**

Algorithm 4: Release Processor
Input: A user selects a processor, presses "release"
Requirements: Unlock a processor.
1: **if** processor owner ≠ self **then** do nothing;
2: **else**
3: **begin transaction**
4: set processor.lock_flag ← 0
5: **for** each outgoing data link
6: **call** release_data_link
7: **end transaction**
8: **endif**

Algorithm 5: Lock Data Link
Input: A user selects a data link and presses "lock"
Requirements: Lock a data link.
1: **if** data link ∉ locked_datalink_list **then** insert
2: **if** data link.lock_flag = 1 **then return** false
3: **else** data_link.owner ← self; lockFlag ← 1
4: **endif**

Algorithm 6: Release Data Link
Input: A user selects a data link, presses "release"
Requirements: Unlock a data link.
1: **if** data link owner ≠ self **then** do nothing;
2: **else** set data_link.lock_flag ← 0
3: **endif**

5.3 Collaboration Transactions

Our locking algorithms facilitate concurrent workflow composition. Actions by each user are modeled as transactions to ensure concurrency control. We further define four basic actions (in the Taverna context): 1) insert a data link, 2) delete a data link, 3) insert a processor, and 4) delete a processor. An update action can be modeled as a delete followed by an insert action. Thus, all collaborative composition actions can be mapped to database update operations. As a result, we can exploit the concurrency control facility of database management systems to ensure the serializability of all executions. Bad transactions will be automatically aborted. We are also working on an exception handling facility; which is out of the scope of this paper. After a user update is successfully committed, all collaborators will be notified, so that each collaborator can have the most up-to-date workflow.

We studied various concurrency control schemes at the database level, aiming to better support collaborative workflow composition. Specifically, we have observed that scientists tend to adopt a "long-thinking" pattern, meaning that they take a long time to think before they actually make changes that take much less amount of time. As a result, the traditional read and write locks may lower the parallelism in such a working style.

Therefore, instead of placing a traditional exclusive write lock on a task, we realized a *Read Committed with first-committer-win (RC_fcw)* scheme, which is an extension of READ COMMITTED with the first-committer-win feature at the SNAPSHOT isolation level. Correctness proofs of RC_fcw can be found in our previous report [52]. Here we discuss how we apply RC_fcw to facilitate collaborative scientific workflow composition.

Definition 1: A provenance log is a tuple L = <D, T, $\sum$, S, $\prod$), where D is data item set, T is transaction set, $\sum$ is atomic operation set, S is the access function that returns the set of items accessed by an atomic operation, and $\prod$ is the permutation function that assigns a sequence number to an operation.

Definition 2: L is RC_fcw *iff* it can be generated by a locking sequence where: 1) a transaction must obtain a write lock on an item before writing it and the write lock is released when the transaction terminates (long-term write lock); 2) a transaction must obtain a read lock on an item before reading it and the read lock is released right after the read operation completes (i.e., a transaction can read only committed values); 3) when a transaction attempts to acquire a lock on an item on which a conflicting lock is held, the transaction will be put into a waiting queue; and 4) after a transaction T_1 reads a data item and before T_1 attempts to write the item, if another transaction T_2 writes the item and commits, T_1 will be aborted (first committer win).

Only the scientist who submits the changes first will get through. As shown in the pseudo code in Fig. 9, RC_fcw is implemented by attaching a version number *vn(x)* to an object x, which is a *synchronization area* as introduced in Section 5.1. *vn(x)* is incremented by one whenever x is updated, i.e., some actions are performed over area x. If a transaction T_1 attempts to update x, T_1 will first read the item associated with its contemporary *vn(x)* to the local drive and work locally. When time comes and T_1 tries to submit the changes to x, a version comparison is triggered to check whether any other transaction has updated x in between. If changes have been submitted since T_1 read x, T_1's submission will be aborted. Otherwise, the commitment will become permanent and the *vn(x)* will be incremented and updated to item x. The check and the update together are performed atomically.

```
Algorithm: RC_fcw scheme part 1.
Input: A transaction t(x) intends to work on an item x.
Requirements: Place an intentional lock.
1: add an intentional lock → x
2: collaborators[list] ← t(x)
2: return vn(x)

Algorithm: RC_fcw scheme part 2.
Input: A transaction t(x) intends to submit changes on item x.
Requirements: RC_fcw.
1: begin transaction
2:   get vn(x new)
3:   if (vn(x new) > vn(x))
4:     abort t(x)
5:   else
6:     vn(x) ← vn(x) + 1
7:     commit t(x)
8:     notify (collaborators[list])
9:   endif
```

Fig. 9. RC_fcw scheme algorithm.

6 System Implementation and Experiments

6.1 System Implementation

We have constructed a collaboration pattern template library. The basic building blocks are collaboration primitives. Users can build new collaboration patterns using existing collaboration primitives. Identified collaboration patterns are stored as provenance data to support the tracking, storing, and querying of interactions and coordination among scientists.

Without reinventing the wheel, we extended the single user-version Taverna into a collaborative version [50]. The reason why we chose Taverna is mainly based on its popularity and large user base [11]. Another reason is that Taverna is an open-source tool developed in Java. Thus we can explore its code base.

We built a central server supporting all workflow collaborations. Workflow evolution provenance and collaboration provenance are stored in a shared database on the server. Each collaborator may store an intermediate version of the workflow at a local machine, but all committed activities are stored at the server, to support asynchronous collaboration where collaborators may work on the shared workflow at preferable time. We consider four options for selecting database systems: native XML, relational, XML-relational, and RDF. Currently we use a relational database because it is a preferable choice of Taverna, upon which our Confucius is built.

Fig. 10 shows a snapshot of our Confucius system sup-

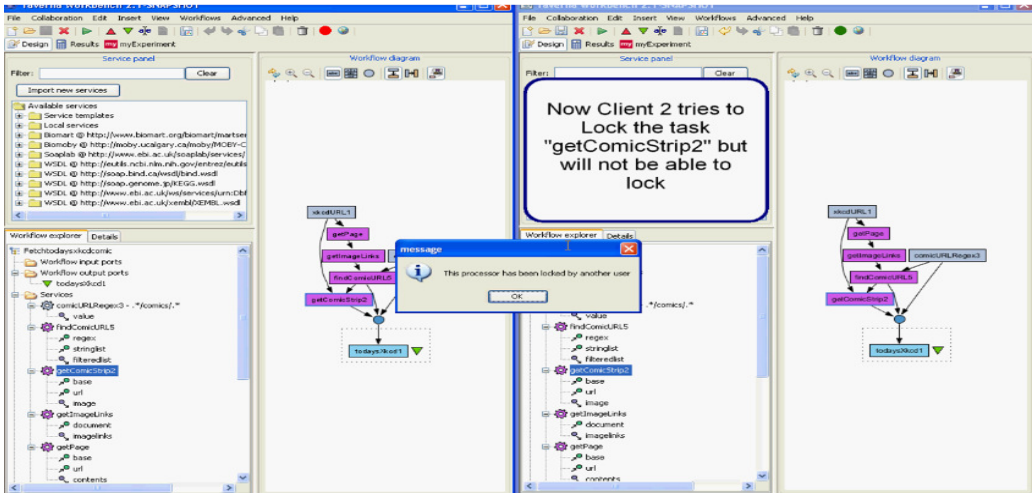

Fig. 10. Screen shots of concurrent workflow updates.

porting concurrent workflow composition. To ease illustration, we show two screens (left and right) representing two scientists running two client versions of Confucius on two distributed machines. Here we use the remote desktop feature of Windows to show the two screens together. Any change (adding or removal of components) in the shared workflow made by one scientist will be instantaneously shown on all collaborators' screens. Shared workflow product is stored at the server, so other collaborators may join the collaboration at any time and review the current workflow if proprietary access control allows. When a scientist applies a write lock on a task on the shared workflow, the other scientist cannot update the task due to our concurrency control policy.

Fig. 11 shows portions of our demo, where role-based P2P collaboration is realized using the centralized server mode, as discussed in Section 3.2. As shown in the upper part of Fig. 11, we added a menu item group "Collaboration" in the menu bar, which supports five actions regarding P2P collaboration: (1) share workflow (a coordinator initiates a shared scientific workflow document), (2) connect (the coordinator allows identified participants to join), (3) disconnect (the coordinator removes a participant from the collaboration), (4) request token (request a floor to have write access), and (5) release token (release the write access of the shared workflow).

The left screen in Fig. 11 shows a scientist who starts a collaboration session. Once the scientist clicks the "Share Workflow" menu item, the collaboration will begin. As highlighted in Fig. 11, the initiator of the collaboration automatically obtains the token (floor), shown in green. She can also click "Release Token" to release the token; and her status will turn back into red by doing so. After a collaboration session is started, other scientists (upon invitations) will be able to select the "Connect" menu item to join the collaboration, and will instantaneously view the same workflow shown on the token holder's screen. As shown in Fig. 11, any collaborator can click "Request Token" to ask for the write privilege. If available, the token will be granted to the requestor.

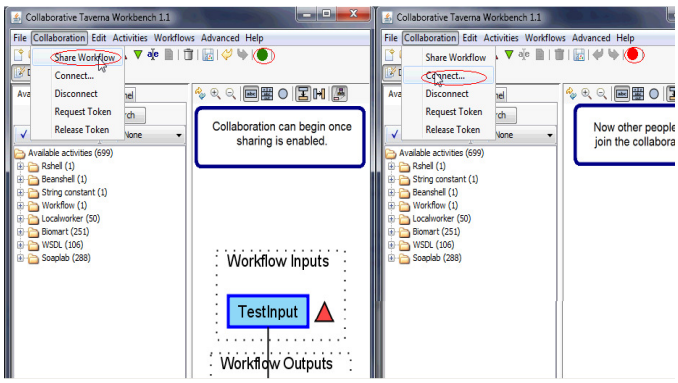

Fig. 11. P2P collaboration.

Fig. 12 shows a portion of a screen shot illustrating that a backdoor communication is initiated between two Confucius instances. A user identifies a specific IP address (i.e., 127.0.0.1) to invite a team member to start a backdoor communication. Our current version offers six functions supporting backdoor communication, as shown in the drop down menu at the upper right corner of Fig. 12: (1) backdoor connection (initiate a backdoor communication session), (2) share workflow (manually enable workflow sharing between backdoor communication participants), (3) connect (invite an additional participant to the backdoor communication), (4) disconnect (remove a participant from the backdoor communication), (5) request token (a participant asks for the mutual exclusive floor for writing access to the shared workflow), and (6) release token (a participant releases the floor to allow other participants to request the floor).

6.2 Experiments

We have designed and conducted a series of experiments to evaluate the performance of our concurrency control scheme (the RC_fcw algorithm) implemented in the Confucius system, in supporting collaborative (i.e., concurrent) workflow composition among a group of scientists. We compared our algorithm with two popular concurrency control schemes: *strict two-phase locking (2PL_wait0 or 2PL in short)* and *strict two-phase locking with update lock (2PL_update)* [53]. Both schemes are extensions of the standard strict two-phase locking scheme to handle deadlocks, while 2PL resolves deadlocks and 2PL_update pre-

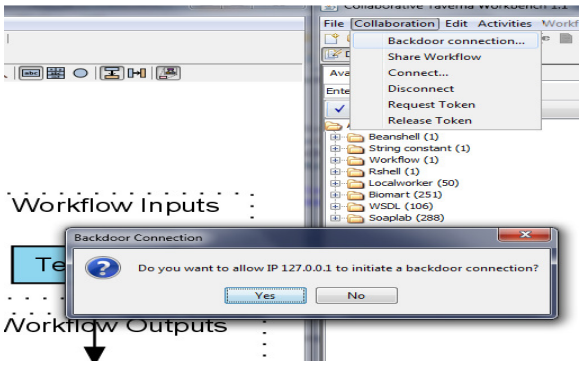

Fig. 12. Backdoor collaboration setting.

vents deadlocks.

Definition 3: L is 2PL_wait0 *iff* it can be generated by a locking sequence where: 1) a transaction must hold a read lock (respectively a write lock) on an item before reading (respectively writing) the item; 2) if a transaction T tries to put a lock on an item on which a conflicting lock is held, T will be aborted immediately (wait 0); and 3) a transaction T holds all obtained locks until it terminates, at that time all locks that T has acquired will be released (strict).

Definition 4: L is 2PL_update *iff* it can be generated by a locking sequence where: 1) a transaction must hold a read lock (respectively a write lock) on an item before reading (respectively writing) the item; 2) if a transaction T attempts to read an item and then possibly modify it later, then T has to acquire an update lock first and then upgrade the update lock to a write lock right before the write operation; 3) if a transaction attempts to obtain an update lock on an item with update locks, it will be put into a waiting queue.

6.2.1 Experimental setup

We simulated a collaborative workflow composition environment, where a group of scientists collaboratively (concurrently) compose (update) different parts of a mutual workflow during a time period. We developed a workflow generator, which can produce a randomly generated scientific workflow comprising a configurable number of tasks. For simplicity, we made the following two assumptions. (1) A workflow under test contains a configurable number of individual tasks, meaning that they do not depend on each other. (2) The concurrency control granularity was set at the task level.

Each collaborator was simulated by an independent (Java) thread, which iteratively reads a random task of the workflow, waits for a predefined time period (e.g., 0.05 seconds for thinking), and then performs an update on the task. The time interval between iterations was set to zero at the moment, but could be configured to other values. To simulate the long-thinking, short-read pattern, the reading and writing time for each thread are neglected. While each collaborator infinitely performed such iterative random updates, we recorded the total numbers of both successful and unsuccessful task updates (due to abort), respectively, within a predefined time window. All experiments were conducted on a PC with Intel Core 2 Duo CPU P8800 (@2.66 GHz & 2.76GHz) and 3 GB main memory, running the Windows 7 Professional Edition operating system. The database system used is Apache Derby 10.5.3.0. The database was installed in an embedded fashion for this experiment, so that data transportation time could be neglected.

Our first set of experiments focused on testing the throughput of the three schemes by varying the number of collaborators. The throughput (collaboration productivity) is defined as the number of successful task updates by all collaborators per minute. The average throughput is calculated for each collaboration group of size N (1, 5, 10, 20, 30, …, 100). To avoid coincidence, for each group size, the experiment was repeated 3 times with the average value calculated. We also monitored the number of failed task updates performed per minute to examine the trend of update conflicts as the number of collaborators increased. Table II summarized the parameters of our experiments.

Table II. Experimental settings.

Number of tasks	20
Number of collaborators	1 to 150
Experiment time	60 seconds
Thinking time	0.05 seconds

6.2.2 Experimental results and discussions

6.2.2.1 Throughput study. Fig. 13 shows the comparison of throughput by varying the number of collaborators for the three schemes. All three schemes show similar productivity when the group size is smaller than 10. However, when the group size grows, their throughputs become significantly different. This phenomenon is caused by the increase of possibility of conflict when the group size grows.

For the RC_fcw scheme, the collaboration productivity steadily increases as the number of collaborators increases, reaching a maximum rate of 11,882 updates per minute at a group size of 40. Afterwards, the throughput starts to decline due to the increase of the number of conflicts that lead to abortion. For the 2PL scheme, the throughput is increased until the group size reaches 20 with the maximum rate of 6,721. Afterwards, the throughput keeps on declining until reaching a very low level, mainly due to the increasing numbers of deadlocks caused by the strict two-phase locking algorithm. For the 2PL_update scheme, the throughput increases monotonically. Especially, the group productivity grows rapidly to reach a throughput of 16,000 at the group size of 60. Afterwards, the group productivity keeps on growing but with a much lower increase rate. Since collaborators lock

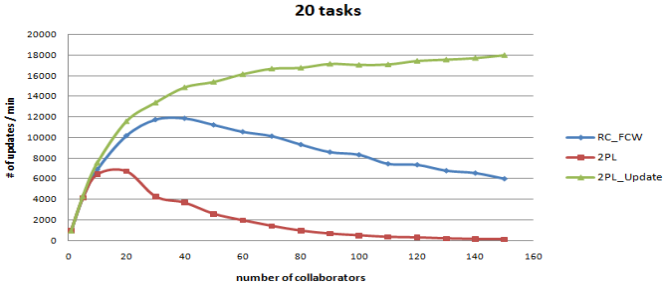

Fig. 13. Comparison of successful update rates.

the tasks before reading them in the 2PL_update scheme, more and more collaborators have to wait in queues when the group size increases.

For the 2PL_update scheme, every collaborator will put an update lock on the task that she is reading, which means other people will not be able to update the task until the lock is released. As the group size grows, the ratio of time for reading and updating of a single task will increase, because the maximum number of tasks that can be simultaneously updated is predefined (e.g., 20). Therefore, eventually productivity will stop increasing.

6.2.2.2 Occupancy study. We monitored the efficiency for each scheme, which is counted as the ratio of a task being occupied on average in the process of a testing time period. The formula is as follows:

$$r = \frac{throughput * unit\_time}{\sum_{i=1}^{task\_no} unit\_time * task\_no}$$

where *throughput* is the total number of successful updates, *unit_time* is the execution time for each collaborator, *task_no* is the total number of tasks in the workflow.

As an example in Fig. 13, the efficiency of 2PL when the number of collaborators is 20 can be calculated as follows:

7566 * 0.05 / (60 * 20) = 31.5%

When the group size is 40, the efficiency of 2PL_update is 61.9%. When the group size reaches 100, its value goes up to 71%. This means that as the number of collaborators increases, more and more tasks will be used at the same time in 2PL_update, which is why its throughput increases monotonically. However, it is also noticeable that the curve of 2PL_update becomes flatter as the number of conflicts increases, since every collaborator will need more time to find an available task to update.

For the RC_fcw scheme, in the beginning, the number of collaborators is less than the number of tasks, so more collaborators will contribute to a higher productivity. As the group size increases, the possibility of multiple collaborators accessing the same task will increase. More users will read the same task, make some changes, and then submit the request. If one collaborator commits, all other collaborators will have to abort their work. On the other hand, in the 2PL_update scheme, a task will be queued when someone is trying to update it. That is why the 2PL_update scheme can show a higher productivity than RC_fcw as the group size increases.

For 2PL, since it uses the SERIALIZABLE isolation level, SELECT statement will get a shared lock on a range of rows; it is highly possible to create a deadlock if two collaborators update tasks at the same time. That's why the throughput of 2PL drops significantly as the group size increases.

6.2.2.3 Failed update rate study. Fig. 14 shows the failed task update rates by varying the number of collaborators for the three schemes.

For the RC_fcw scheme, as the number of collaborators increases, the number of conflicts and hence the number of failed task updates also increase. Note that RC_fcw also has an increasing collaboration productivity before reaching 40 collaborators. This means that an increasing number of failed task updates is more than compensated by the increased number of successful task updates.

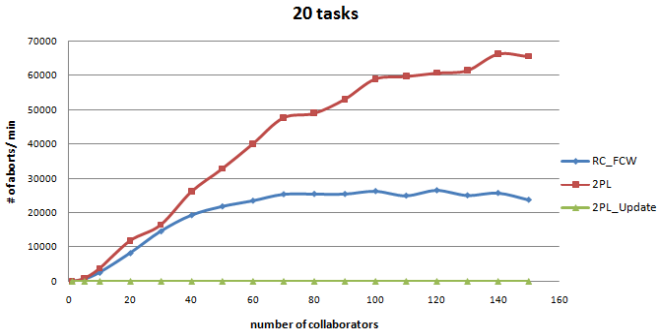

Fig. 14. Failed task update rates under numbers of collaborators.

However, when the group size exceeds 40, the number of conflicts decreases the productivity of every collaborator. One reason may be a scientist may find that another collaborator has already updated the task, which forces her to abort the task.

For the 2PL scheme, the number of abortions significantly increases as the number of collaborators increases. Because of its strict two-phase locks, more conflicts cause more deadlocks that lead to more abortions.

For the 2PL_update scheme, because of the use of update locks, no collaborator could get the resource if there is any conflict, so no abortion will occur.

6.2.2.4 Analysis. From this comparison, it can be observed that RC_fcw shows similar throughput to 2PL_update for a smaller group size (less than 20 collaborators).

Although the 2PL_update scheme yields the highest throughput for a larger group of collaborators, one issue is significant. Collaborators cannot access a task even though the person who is reading the task does not intend to update it. It is possible that many collaborators want to read a task, but only a few of them want to modify it. Under such a circumstance, RC_fcw remains a good option because it allows multiple users to read a common task at the same time.

6.2.2.5 Scalability study. We further studied the scalability of RC_fcw in the Confucius system under different numbers of tasks and collaborators. We chose three fixed numbers for collaborators (2, 10 and 50) to represent small, medium, and large collaborator groups, respectively. The total number of tasks was set from 10 to 100, with 10 as the incremental step. For each testing number of tasks, the experiment was repeatedly performed three times with the average value calculated. Table III summarizes the setting parameters of our experiments.

Table III. Experimental Settings.

Number of collaborators	2, 10, 50
Number of tasks	10 to 100
Experiment time period	60 seconds
Thinking time	0.05 second

Fig. 15 shows our experimental results: (a)~(c) show successful task updates; (d)~(f) show failed task updates.

In Fig. 15(a), two collaborators keep updating tasks, so the possibility that they intend to update the same task drops as the number of tasks increases. In the beginning, 2PL_update has more successful updates than the other two when conflicts exist. The RC_fcw curve stays slightly below the other two curves, indicating that its performance is lower when the conflict level is low. The reason might be the overhead inherent in the RC_fcw scheme, as it has to keep on identifying the first committer.

In Fig. 15(b), 10 collaborators work together. With a higher conflict rate, the rank of throughput is: 2PL < RC_fcw < 2PL_update. After 40 tasks, the 2PL curve gradually exceeds the RC_fcw curve, because the number of conflicts drops as the number of tasks increases and the overhead of RC_fcw gradually dominates. All three curves become flat when the number of tasks exceeds 50, since the conflict level becomes very low.

In Fig. 15(c), 50 collaborators work together. Under a medium to high level of conflict, the rank of throughput is: 2PL_update > RC_fcw > 2PL. The collaboration productivity in all three schemes increases monotonically when the workflow comprises less than 100 tasks.

Fig. 15(a)~(c) indicate that 2PL and 2PL_update show similar performance at the low conflict level, because their locks behave similarly when there is no conflict.

Fig 15(d)~(f) show failed update rates with the same experimental settings. RC_fcw constantly show lower failure rates than 2PL.

In summary, our scalability experiments proved that our RC_fcw scheme provides decent throughout and failure rates for different sizes of collaboration groups.

7 Conclusions

In this paper, we presented our ongoing work on establishing collaboration protocols to support collaborative scientific workflow composition. Our service-oriented infrastructure includes a collaboration ontology associated with a set of collaboration patterns, primitives, and constructs, as well as concurrent control mechanisms to support concurrent collaborative workflow composition.

We plan to continue our research in the following directions. First, we will design and conduct an evaluation study and use the feedback to enhance the system. Second, based on the collaboration ontology, we plan to enhance collaboration provenance management performance. Third, we plan to conduct more experiments to study the effects of tuning various parameters (e.g., the number of concurrent collaborators, the productivity of individual members, the number of tasks comprised in the shared scientific workflow) on concurrent productivity. Fourth, we plan to explore conducting collaborative scientific workflow composition in the Cloud infrastructure.

Acknowledgment

The authors thank Sha Liu for the assistance in the presented experimental study. This work is supported by National Science Foundation, under grants NSF IIS-0959215 and IIS-0960014.

References

[1] G.M. Olson, A. Zimmerman, and N. Bos, *eds., Scientific Collaboration on the Internet*, 2008, MIT Press, Cambridge, MA, USA.

[2] LSST, "Large Synoptic Survey Telescope", 2009, Accessed on, Available from: http://www.lsst.org/lsst/science.

[3] B. Ludäscher, "Scientific Workflows: Cyberinfrastructure for e-Science", in Proceedings of *Pacific Neighborhood Consortium (PNC)*, 2007, Berkeley, CA, USA, Oct. 19, pp.

[4] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit, "Artificial Intelligence and Grids: Workflow Planning and Beyond", *IEEE Intelligent Systems*, Jan.-Feb., 2004, 19(1): pp. 26–33.

[5] E. Deelman and Y. Gil, "NSF Workshop on the Challenges of Scientific Workflows", (ed.), May 1-2, 2006.

[6] S. Wuchty, B. Jones, and B. Uzzi, "The Increasing Dominance of Teams in Production of Knowledge", *Science*, 2007, 316: pp. 1036-1039.

[7] N.R. Council, "Facilitating Interdisciplinary Research". 2004, National Academies Press, Washington DC, USA.

[8] G. Bell, T. Hey, and A. Szalay, "Beyond the Data Deluge", *Science*, 2009, 323(5919): pp. 1297-1298.

[9] S. Lu and J. Zhang, "Collaborative Scientific Workflows Supporting Collaborative Science", *International Journal of Business Process Integration and Management (IJBPIM)*, 2011, 5(2): pp. 185-199.

[10] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, and Y. Zhao, "Scientific Workflow

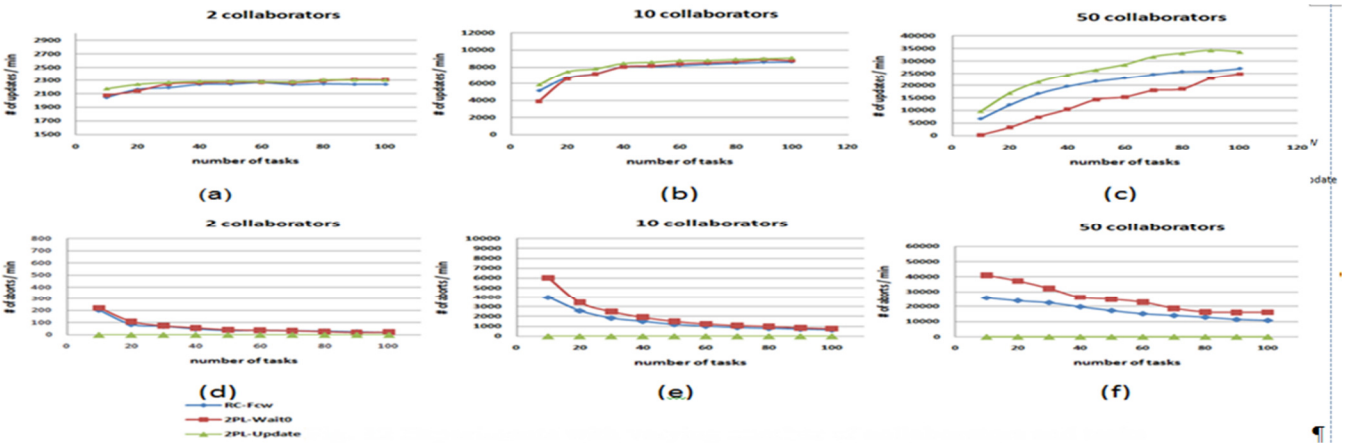

Fig. 15. Comparison of successful update rates.

Management and the Kepler System", *Concurrency and Computation: Practice & Experience*, 2006, 18(10): pp. 1039-1065.
[11] T. Oinn, M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M.R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: Lessons in Creating a Workflow Environment for the Life Sciences", *Concurrency and Computation: Practice & Experience*, 2006, 18(10): pp. 1067–1100.
[12] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang, "Programming Scientific and Distributed Workflow with Triana Services", *Concurrency and Computation: Practice & Experience*, 2006, 18(10): pp. 1021–1037.
[13] J. Freire, C.T. Silva, S.P. Callahan, E. Santos, and C.E. Scheidegger, "Managing Rapidly-Evolving Scientific Workflows ", *Lecture Notes in Computer Science*, May, 2006, 4145/2006: pp. 10–18.
[14] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, Reliable, Loosely Coupled Parallel Computation", in Proceedings of *IEEE International Workshop on Scientific Workflows*, 2007, Salt Lake City, UT, USA, Jul. 9-13, pp. 199–206.
[15] A. Chebotko, C. Lin, X. Fei, Z. Lai, S. Lu, J. Hua, and F. Fotouhi, "VIEW: A Visual Scientific Workflow Management System", in Proceedings of *the 1st IEEE International Workshop on Scientific Workflows*, 2007, Salt Lake City, UT, USA, Jul., pp. 207–208.
[16] C. Lin, S. Lu, Z. Lai, A. Chebotko, X. Fei, J. Hua, and F. Fotouhi, "Service-Oriented Architecture for VIEW: A Visual Scientific Workflow Management System", in Proceedings of *the IEEE 2008 International Conference on Services Computing (SCC)*, 2008, Honolulu, HI, USA, Jul. 9-11, pp. 335-342.
[17] D.D. Roure, C. Goble, and R. Stevens, "The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows", *Future Generation Computer Systems*, 2009, 25: pp. 561-567.
[18] L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*, 2007, Springer.
[19] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, and Y. Simmhan, "The Trident Scientific Workflow Workbench", in Proceedings of *4th IEEE International Conference on eScience*, 2008, Dec. 7-12, pp. 317-318.
[20] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi, "Storing and Querying Scientific Workflow Provenance Metadata Using an RDBMS", in Proceedings of *the 3rd IEEE International Conference on e-Science and Grid Computing*, 2007, Bangalore, India, Dec. 10-13, pp. 611–618.
[21] A. Chebotko, S. Lu, X. Fei, and F. Fotouhi, "RDFProv: A Relational RDF Store for Querying and Managing Scientific Workflow Provenance", *Data & Knowledge Engineering (DKE)*, 2010, 69(8): pp. 836-865.
[22] A. Chebotko, S. Lu, and F. Fotouhi, "Semantics Preserving SPARQL-to-SQL Query Translation", *Data & Knowledge Engineering*, 2009, 68(10): pp. 973-1000.
[23] S. Lu and J. Zhang, "Collaborative Scientific Workflows", in Proceedings of *IEEE International Conference on Web Services (ICWS)*, 2009, Los Angeles, CA, USA, Jul. 6-10, pp. 527-534.
[24] G. Fakas and B. Karakostas, "A Workflow Management System Based on Intelligent Collaborative Objects", *Information & Software Technology*, 1999, 41(13): pp. 907-915.
[25] H. Song, J.J. Dong, C. Han, W.R. Jung, and C.-H. Youn, "A SLA-Adaptive Workflow Integrated Grid Resource Management System for Collaborative Healthcare Services", in Proceedings of *the 3rd International Conference on Internet and Web Applications and Services (ICIW)*, 2008, Athens, Greece, Jun. 8-13, pp. 702-707.
[26] L. Pudhota and E. Chang, "Collaborative Workflow Management Using Service Oriented Approach", in Proceedings of *International Conference on E-Business, Enterprise Information Systems, E-Government (EEE)*, 2005, Las Vegas, USA, Jun. 20, pp. 167-173.
[27] R. Lu and S.W. Sadiq, "A Survey of Comparative Business Process Modeling Approaches", in Proceedings of *the 10th International Conference on Business Information Systems (BIS)*, 2007, Poznan, Poland, Apr. 25-27, pp. 82-94.
[28] C.-J. Huang, C.V. Trappey, and C.C. Ku, "A JADE-Based Autonomous Workflow Management System for Collaborative IC Design", in Proceedings of *the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2007, Melbourne, Australia, Apr. 26-28, pp. 777-782.
[29] J. Dang, J. Huang, and M.N. Huhns, "Workflow Coordination for Service-Oriented Multiagent Systems", in Proceedings of *the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2007, pp. 1056-1058.
[30] J. Balasooriya, S.K. Prasad, and S.B. Navathe, "A Middleware Architecture for Enhancing Web Services Infrastructure for Distributed Coordination of Workflows", in Proceedings of *IEEE International Conference on Services Computing (SCC)*, 2008, Jul., pp. 370-377.
[31] J. Balasooriya, J. Joshi, S.K. Prasad, and S. Navathe, "A Two-Layered Software Architecture for Distributed Workflow Coordination over Web Services", in Proceedings of *IEEE International Conference on Web Services (ICWS)*, 2006, Sep., pp. 933-934.
[32] P. Kazanis and A. Ginige, "Asynchronous collaborative business process modeling through a web forum", in Proceedings of *Seventh Annual CollECTeR Conference on Electronic Commerce*, 2002, Melbourne, VIC, Australia, pp.
[33] T. Miller, P. McBurney, J. McGinnis, and K. Stathis, "First-Class Protocols for Agent-Based Coordination of Scientific Instruments", in Proceedings of *16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2007, Jun., pp. 41-46.
[34] Z. Nemeth, C. Perez, and T. Priol, "Distributed Workflow Coordination: Molecules and Reactions", in Proceedings of *20th IEEE International Parallel & Distributed Processing Symposium*, 2006, Apr., pp. 260-267.
[35] C. Yen, W.J. Li, and J.C. Lin, "A web-based collaborative, computer-aided sequential control design tool", *IEEE Control Systems Magazine*, Apr., 2003, 23(2): pp. 14-19.
[36] D. Dori, D. Beimel, and E. Toch, "OPCATeam - Collaborative Business Process Modeling with OPM", in Proceedings of *2nd International Conference on Business Process Management (BPM)*, 2004, Potsdam, Germany, Jun. 17-18, Springer-Verlag Berlin Heidelberg, pp. 66-81.
[37] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke, "OntoEdit: collaborative ontology engineering for the semantic Web", in Proceedings of *the First International Semantic Web Conference 2002 (ISWC), LNCS 2342*, 2002, Springer, pp. 221-235.
[38] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller, "WS-BPEL Extension for People (BPEL4People), Version 1.0", 2007 Jun., Accessed on, Available from: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf.
[39] N. Ayachitula, M.J. Buco, Y. Diao, M. Surendra, R. Pavuluri, L. Shwartz, and C. Ward, "IT Service Management Automation - A Hybrid Methodology to Integrate and Orchestrate Collaborative Human Centric and Automation Centric Workflows", in Proceedings of *IEEE International Conference on Services Computing (SCC)*, 2007, Salt Lake City, UT, USA, Jul. 9-13, pp. 574-581.

[40] D. Russell, P.M. Dew, and K. Djemame, "Service-Based Collaborative Workflow for DAME", in Proceedings of *IEEE International Conference on Services Computing (SCC)*, 2005, Orlando, FL, USA, Jul. 11-15, pp. 139-146.
[41] D. Jordan and J. Evdemon, "Web Services Business Process Execution Language, Version 2.0", 2007 Apr., Accessed on, Available from: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[42] J.Y. Sayah and L.-J. Zhang, "On-Demand Business Collaboration Enablement with Services Computing", *Decision Support Systems*, Jul., 2005, 40(1): pp. 107-127.
[43] C.K. Chang, J. Zhang, and K.H. Chang, "Survey of Computer Supported Business Collaboration in Support of Business Processes", *International Journal of Business Process Integration and Management (IJBPIM)*, 2006, 1(2): pp. 76-100.
[44] I. Altintas, *Collaborative Provenance for Workflow-Driven Science and Engineering*, Ph.D thesis, Universiteit van Amsterdam, 2011.
[45] A. Chapman, H.V. Jagadish, and P. Ramanan, "Efficient Provenance Storage", in Proceedings of *ACM International Conference on Management of Data (SIGMOD)*, 2008, Vancouver, Canada, Jun. 9-12, pp. 993-1006.
[46] M.K. Anand, S. Bowers, T.M. McPhillips, and B. Ludäscher, "Efficient Provenance Storage over Nested Data Collections", in Proceedings of *EDBT*, 2009, pp. 958-969.
[47] T. Oinn, "XScufl Language Reference", 2004, Accessed on, Available from: http://www.ebi.ac.uk/~tmo/mygrid/XScuflSpecification.html.
[48] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995, Addison Wesley, Boston, MA, USA.
[49] M. Robert, W.J. Evans, D.H. Honemann, and T.J. Balch, *Robert's Rules of Order, Newly Revised, 10th Edition*, 2000, Perseus Publishing Company.
[50] J. Zhang, "Co-Taverna: A Tool Supporting Collaborative Scientific Workflows", in Proceedings of *IEEE International Conference on Services Computing (SCC)*, 2010, Miami, FL, USA, Jul. 5-10, pp. 41-48.
[51] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", in Proceedings of *OSDI*, 2004, pp. 137–150.
[52] S. Lu, A. Bernstein, and P. Lewish, "Correct Execution of Transactions at Different Isolation Levels", *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, September, 2004, 16(9): pp. 1070-1081.
[53] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques (1st edition)*, 1992, Morgan Kaufmann.

Jia Zhang, Ph.D., is an Associate Professor of Department of Computer Science at Northern Illinois University. Her current research interests center on Services Computing, with a focus on Internet-centric collaboration, service-oriented architecture, and semantic service discovery. She has co-authored 1 book and published over 120 journal articles, book chapters, and conference papers. Zhang is an associate editor of IEEE Transactions on Services Computing (TSC) and International Journal on Web Services Research (JWSR). She is a member of the IEEE and can be reached at jia-zhang@cs.niu.edu.

Daniel Kuc is a master student at Department of Computer Science at Northern Illinois University. His research focuses on collaborative scientific workflow composition and service-oriented architecture.

Shiyong Lu, Ph.D., is an Associate Professor of Department of Computer Science at Wayne State University. His current research interests focus on scientific workflows and provenance data management. He has published 2 books and over 100 papers. He is the founding chair and program chair of IEEE International Workshop on Scientific Workflows since 2007. Dr. Lu is an associate editor of the International Journal of Cloud Applications and Computing, and an editorial board member of the International Journal of Healthcare Information Systems and Informatics. He is a Senior Member of the IEEE. He can be reached at shiyong@wayne.edu.