Report Number: wucse-2009-4

2009

Achieving Coordination Through Dynamic Construction of Open Workflows ** PLEASE SEE WUCSE-2009-14 **

Louis Thomas, Justin Luner, Grui-Catalin Roman, and Christopher Gill

Workflows, widely used on the Internet today, typically consist of a graph-like structure that defines the orchestration rules for executing a set of tasks, each of which is matched at run-rime to a corresponding service. The graph is static, specialized directories enable the discovery of services, and the wired infrastructure supports routing of results among tasks. In this paper we introduce a radically new paradigm for workflow construction and execution called open workflow. It is motivated by the growing reliance on wireless ad hoc networks in settings such as emergency response, field hospitals, and military operations. Open workflows facilitate...
Read complete abstract on page 2.

Achieving Coordination Through Dynamic Construction of Open Workflows ** PLEASE SEE WUCSE-2009-14 **

Louis Thomas, Justin Luner, Grui-Catalin Roman, and Christopher Gill

Complete Abstract:

Workflows, widely used on the Internet today, typically consist of a graph-like structure that defines the orchestration rules for executing a set of tasks, each of which is matched at run-rime to a corresponding service. The graph is static, specialized directories enable the discovery of services, and the wired infrastructure supports routing of results among tasks. In this paper we introduce a radically new paradigm for workflow construction and execution called open workflow. It is motivated by the growing reliance on wireless ad hoc networks in settings such as emergency response, field hospitals, and military operations. Open workflows facilitate goal-directed coordination among physically mobile agents (people and host devices) that form a transient community over an ad hoc wireless network. The quintessential feature of the open workflow paradigm is the ability to construct a custom context-specific workflow specification on the fly in response to unpredictable and evolving circumstances by exploiting the knowhow and services available within a given spatiotemporal context. This paper introduces the open workflow approach and explores the technical challenges (algorithms and architecture) associated with its first practical realization.

Achieving Coordination Through Dynamic Construction of Open Workflows ** PLEASE SEE WUCSE-2009-14 **

Authors: Louis Thomas, Justin Luner, Gruia-Catalin Roman, and Christopher Gill

Corresponding Author: thomasl@cse.wustl.edu

Web Page: http://mobilab.cse.wustl.edu/Research/index.htm

Abstract: Workflows, widely used on the Internet today, typically consist of a graph-like structure that defines the orchestration rules for executing a set of tasks, each of which is matched at run-rime to a corresponding service. The graph is static, specialized directories enable the discovery of services, and the wired infrastructure supports routing of results among tasks. In this paper we introduce a radically new paradigm for workflow construction and execution called open workflow. It is motivated by the growing reliance on wireless ad hoc networks in settings such as emergency response, field hospitals, and military operations. Open workflows facilitate goal-directed coordination among physically mobile agents (people and host devices) that form a transient community over an ad hoc wireless network. The quintessential feature of the open workflow paradigm is the ability to construct a custom context-specific workflow specification on the fly in response to unpredictable and evolving circumstances by exploiting the knowhow and services available within a given spatiotemporal context. This paper introduces the open workflow approach and explores the technical challenges (algorithms and architecture) associated with its first practical realization.

Achieving Coordination Through Dynamic Construction of Open Workflows

Louis Thomas, Justin Luner, Gruia-Catalin Roman, and Christopher Gill

Department of Computer Science and Engineering
Washington University in St. Louis
{thomasl,jluner,roman,cdgill}@cse.wustl.edu

Abstract. Workflows, widely used on the Internet today, typically consist of a graph-like structure that defines the orchestration rules for executing a set of tasks, each of which is matched at run-rime to a corresponding service. The graph is static, specialized directories enable the discovery of services, and the wired infrastructure supports routing of results among tasks. In this paper we introduce a radically new paradigm for workflow construction and execution called open workflow. It is motivated by the growing reliance on wireless ad hoc networks in settings such as emergency response, field hospitals, and military operations. Open workflows facilitate goal-directed coordination among physically mobile agents (people and host devices) that form a transient community over an ad hoc wireless network. The quintessential feature of the open workflow paradigm is the ability to construct a custom context-specific workflow specification on the fly in response to unpredictable and evolving circumstances by exploiting the knowhow and services available within a given spatiotemporal context. This paper introduces the open workflow approach and explores the technical challenges (algorithms and architecture) associated with its first practical realization.

1 Introduction

As computing and communication are becoming more and more integrated into the fabric of our society, new kinds of enterprises and new forms of social interactions emerge. Workflow technology, for instance, allows end users to initiate complex goal-oriented activities that leverage off services made available by a wide range of enterprises that have a virtual presence on the Internet. The typical scenario is one in which a user employs a web interface to make a request to a workflow engine responsible for executing a predefined workflow specification that can satisfy the specific user need, e.g., print photos, reserve tickets, or make a bid. The workflow specification takes the form of a directed acyclic graph with vertices denoting tasks and edges defining an execution order along with the flow of data and control. Each task is itself a specification for a service to be discovered and invoked by the workflow engine. What makes the workflow paradigm successful is the high degree of decoupling that it exhibits at multiple levels: between the user need and the workflow required to satisfy it, between the workflow specification and the services available to execute it, and between the service invocations and their implementations.

Current workflow technology relies on servers, which manage service directories, store statically defined workflows, and host the workflow management system or engine. Formal studies have led to a better understanding of the range of constructs needed to support useful workflow specifications [1] and have been complemented by efforts to develop industrial grade workflow specification languages, such as BPEL [2], YAWL [3], WfXML [4], and XLANG [5]. Some researchers also have considered the application of artificial intelligence techniques to automatic construction of workflow specifications [6] from information available in a centralized knowledgebase.

Despite being an established and mature technology, efforts toward using workflow technology in ad hoc wireless environments are relatively new. They include the development of workflow execution engines targeted to small portable devices [7], and techniques for executing workflows in mobile wireless networks [8]. These studies reveal the need for a major reevaluation of the way one thinks about workflow technology: hosts move, service availability depends upon which hosts are within communication range, user needs tend to be situational, and one cannot anticipate the range of responses demanded by changing circumstances. These observations suggest that, in ad hoc wireless settings, it is desirable to tailor or generate workflows dynamically so as to be responsive despite variability in available resources and knowhow.

Starting with this premise, we pose the question of how workflow technology might be reshaped for use in the absence of any wired connectivity. Application domains that satisfy this assumption include low profile military operations, emergency responses to major natural disasters, scientific expeditions in remote parts of the globe, field hospitals, and large construction sites. These application domains share several key features: ad hoc interactions among people, high levels of mobility, the need to respond to unexpected developments, the use of locally available resources, prescribed rules of operation, and specialized knowhow. For instance, consider a construction worker discovering a mercury spill. While there is a prescribed response, it is his supervisor who has the needed expertise and the training. She can initiate the response, but access to the spill is made difficult by a support structure whose dismantling requires some special intervention which only the chief engineer can manage. The result is a series of frantic phone calls and the dispatching of various workers and equipment to execute what might be seen as a workflow that is reactive, opportunistic, composite, and constrained by the set of participants present on the site along with their knowhow and resources. Workflow, whether carried out by people or machines, thus becomes a coordination vehicle for social and business activities. In this paper we explore whether workflow technology can become a coordination mechanism for activities that are carried out in an ad hoc setting.

We use the term *open workflow* to denote a workflow specification, construction, and execution paradigm that is shaped by the dynamics and constraints of an enterprise whose underlying infrastructure is a mobile ad hoc wireless network. We assume a system which evolves over time and consists of a set of participants (host devices and the people who carry them) who move through space and can interact with each other and with a real world environment. Changes in the environment can trigger events that demand a system response that is codified as a dynamically created workflow and is executed by the participants in a distributed manner. The defining feature of the open workflow paradigm is the workflow construction process: workflow fragments codifying individual knowhow distributed across the set of participants are assembled into a custom workflow. In doing so, we also consider the available resources expressed as services offered by the participants along with the mobility of the participants and their willingness to commit to being present at a specific place and at a predetermined time. The latter highlights another feature of the open workflow paradigm, its spatiotemporal dimension.

The technical contributions of this investigation are quite broad. Our earlier efforts to introduce workflow into the wireless ad hoc network setting are extended: the workflow is constructed dynamically in a manner that is sensitive to the availability of resources and is based on knowledge held by colocated participants. A first algorithm for incremental construction of workflows in a dynamic setting is described. In doing so, we have affected a major paradigm shift in the workflow technology. Open workflows are more than sophisticated scripts that enable one to exploit available services — they have become a coordination vehicle that allows cooperating participants to construct and execute a response to a need identified by one of the participants. From a pragmatic point of view, the notion of open workflow enables the development of an entirely new class of enterprise systems that are nimble, mobile, and supportive of a new style of coordination. Finally, it should be noted that the practical importance of this paradigm is enhanced by the tight integration between people, machines, and the physical environment.

The remainder of this paper is organized as follows: Section 2 defines the issues and challenges associated with open workflow construction in dynamic environments. Section 3 explains our algorithm for the collaborative construction of open workflows. Section 4 describes the process of allocation and execution of open workflow tasks in a distributed manner. Section 5 highlights the architecture of an open workflow management system we are currently implementing. Section 6 highlights related research and contrasts it with this work. We provide conclusions in Section 7.

2 Problem Definition

The motivation for this work rests with the idea of creating new coordination modalities appropriate for mobile ad hoc settings. We assume a set of participants who share a common space and sense of purpose. Each community is transient because its members are mobile and it is formed by virtue of the fact that each participant is colocated and has access to wireless communication — for the sake of simplicity one can view a participant as an individual who carries a mobile computing and communication device, which enables the formation of a wireless ad hoc network. In our approach, one of the members of a community identifies some need for action, which results in the automatic construction of a workflow. The workflow is built by combining relevant task graphs[1] known to the community (henceforth called workflow fragments) that encode the collective knowhow available to the community as a whole. Once constructed, the tasks in the workflow are allocated to able participants and the workflow is executed in a distributed and cooperative manner. A participant can commit to executing a particular task in the workflow if they support the service required by that specific task specification at a prescribed location and time and if they can deliver successfully the results to subsequent tasks that require them in order to start execution. Building a workflow on the fly from available contextual knowledge, *i.e.*, the open workflow paradigm, is new and defines the core technical contribution of this paper.

In order to focus on the essential features of the open workflow paradigm, we start with the simplifying assumption that a workflow is a directed acyclic graph in which nodes represent tasks and edges represent data dependencies. A task can execute only when all necessary inputs are available. A *disjunctive* task accepts multiple inputs but needs only one input in order to execute, while a *conjunctive* task needs all of its inputs to execute. Execution of a task consists of the invocation of a service satisfying the respective task specification, and upon completion, a task generates data on all its outputs. As one might expect, the type associated with data carried along each edge has to be compatible with the type expected/generated by the tasks connected by that directed edge. If type labels are attached to each edge, it becomes possible to splice two workflows together in a manner that preserves the consistency of the data flow. This makes workflow composition possible, a necessary requirement for the construction of a new workflow from existing fragments. However, a limiting factor associated with the reliance on edge typing is the fact that many edges may carry similarly typed data in different stages of logical processing — semantic labels are thus needed in order to differentiate identically typed edges referring to different logical results. In our work, composition is based on matching output edges and input edges that carry identical labels.

These observations led us to rethink the workflow as a directed acyclic bipartite graph. Some nodes represent tasks while other nodes denote labels with the additional constraints that (1) all sources (nodes without any incoming edges) and all sinks (nodes without any outgoing edges) are labels, (2) no two nodes carry the same label, and (3) a label can have at most one incoming edge. This definition allows us to compose two workflows by merging (a) identically-named sinks from one workflow with the corresponding sources from the other workflow and (b) identically named sources in both workflows. Two workflows are composable if and only if the result of matching sinks and sources is also a proper workflow, *i.e.*, no two nodes have the same label. For instance, a workflow W_1

[1] For this paper, we assume these are valid workflows in their own right, per the formalization in this section.

with sources $\{a, b, c\}$ and sinks $\{d, e, f\}$ and a workflow W_2 with sources $\{c, d, e\}$ and sinks $\{g, h\}$ can be composed into a new workflow W with sources $\{a, b, c\}$ and sinks $\{f, g, h\}$.

At the abstract level, we build a new workflow on the fly by composing workflow fragments that represent local knowledge held by various participants. This is a goal-oriented process that demands a formal specification as its starting point. Because of the semantics that we presume are associated with the labels, it possible to think of a workflow as a semantic transformation captured by the labels associated with its sources and sinks, henceforth called the *inset* and the *outset* of the workflow. We can further generalize this notion by characterizing a workflow in terms of properties of its inset and outset. We can say for instance that a workflow W with inset $W.in$ and outset $W.out$ satisfies a specification S if and only if $S(W.in, W.out)$ holds, where

$$S \in \mathcal{P}(\mathit{Labels}) \times \mathcal{P}(\mathit{Labels}) \mapsto \mathit{Boolean}$$

When convenient, $S(W)$ may be used as a shorthand for $S(W.in, W.out)$. The definition is general enough so as to allow us to express a wide range of specifications. For instance, a specification of the form

$$S(W) = (W.in = \text{‘\textit{injured person}’})$$

may be satisfied by a workflow whose source is labeled ‘injured person’ and whose sink is labeled ‘hospitalized person,’ ‘treated person,’ or anything else.

Composing workflow fragments may produce a workflow that cannot satisfy a specification S only because it has extra sinks or sources. We can prune a workflow to remove unnecessary data flows, subject to these constraints which ensure the result remains a proper workflow: (1) task outputs that are sinks can be pruned so long as every task has at least one output, (2) task inputs that are sources can be pruned for disjunctive tasks so long as every task has at least one input, and (3) tasks can be pruned so long as any task inputs that are sources and task outputs that are sinks are also pruned. For instance, a workflow W consisting of one disjunctive task t_1 having sources $\{a, b, c\}$ and sinks $\{d, e\}$ can be pruned to produce the workflow W' containing task t_1 with source $\{b\}$ and sink $\{d\}$.

The open workflow construction problem can be defined now as follows. Given a workflow specification S and a set of workflow fragments KW, find a set of workflow fragments in KW, which may be composed (subject to pruning) into a workflow W that satisfies S — we say that W is feasible given S and KW. It is important to note that the defining features of the open workflow paradigm rest with the fact that the specification S can be generated dynamically in response to a situation on the ground and that the set KW represents the combined knowledge of the community as a whole. KW is distributed and dynamic. Different members of the community may carry different workflow fragments that capture knowhow about how to perform specific activities, given certain assumptions expressed by the labels in the inset and guaranteed to achieve certain goals captured by the labels in the outset. As participants move around in space, the knowledge available to the community changes with its membership. For the same specifications, different communities may respond differently or may be unable to construct an appropriate workflow.

Even when a workflow satisfying the desired specification can be constructed, one cannot guarantee that the needed services will be available at the prescribed time and the required location and that reliable communication of the results among various tasks is possible. As such, a practical solution to the open workflow problem requires three basic properties: the workflow being constructed must be *feasible*, *allocatable*, and *executable*. A workflow can be allocated to a community if each task can be serviced at the prescribed time and location. To make this possible we assume that the mobile participants who are able to provide the services required by specific tasks are also willing to commit to being at the right location at the right time. A workflow is executable if the participants supporting the execution of tasks that depend upon each other are able to communicate the needed results in a timely fashion. This is easy to guarantee if the community is stable and all participants are mutually reachable. More sophisticated routing techniques and analysis [9] may be needed if the movement of participants results in temporary disconnections.

3 Collaborative Open Workflow Construction

In this section we introduce our first algorithm for constructing an open workflow. As before we assume KW is the set of all workflow fragments known to the community and S is the specification we wish to satisfy. For the purpose of illustration, we start with the simplifying assumption that S is of the form

$$W.in \subseteq \iota \wedge W.out = \omega$$

where ι and ω are sets of labels with ι being the labels that represent the triggering conditions and ω being the labels that represent the goal.

The approach we take is to assume a participant has identified a need for action and generated a specification S of this form. The participant is in contact with the other members of a community and collects from each a set of workflow fragments which it combines to create the set KW. Using the gathered information, the participant runs our algorithm to find a feasible workflow — a workflow composed of fragments from KW (subject to pruning) that satisfies S — if one exists. We only consider here the issue of generating one feasible workflow, although there are potentially many ways of combining fragments in KW to satisfy S.

Our general strategy is to initially combine all workflow fragments from KW into one large graph, henceforth called the workflow supergraph G. The supergraph represents a unified view of all possible actions represented in the set KW, however it is not necessarily a valid workflow since it may have cycles, outputs produced by multiple tasks, unavailable inputs, or undesirable outputs. We use a node coloring process on the supergraph G to identify one feasible workflow within this graph. We start by coloring the nodes corresponding to set ι of the specification S. Following the data flows, we explore the graph, growing the colored section as we identify which tasks and labels are reachable from ι. We call a label reachable when it is in ι or when it denotes the output of a reachable task; a task is reachable when all necessary input labels are available for its execution via some path starting from ι.

Once we have reached all the elements of set ω, we turn around and begin to prune the reachable set down to a valid workflow. Working backwards with a new color, we identify only those paths which are actually required to reach ω. The pruning phase removes cycles, ensures only one task produces each output, and excludes undesirable outputs. Once the second color has swept all the way back to ι, we have fully identified W, a valid workflow that satisfies specification S and is composed only of fragments in KW subject to some additional pruning of outputs and unneeded paths.

With the general strategy in mind, we now delve into the specifics of how each step is accomplished. Once the specification S has been created and the set of fragments KW collected, we must create the supergraph. We combine the fragments to form the workflow supergraph G as follows: starting with an empty graph, every node and edge of every fragment in KW is considered sequentially and added to G if that node or edge does not already exist in G. When this process is complete, every fragment in KW is a subgraph of G and there are no duplicate nodes or edges in G.

The supergraph G we constructed may have cycles, but cycles are not allowed in our final workflow. During the forward exploration phase, cycles are easily detected as producing a dead end path reaching only nodes already colored by that phase. During the backward pruning phase, cycles are not detectable with simple graph coloring because they are indistinguishable from the normal merging of two paths. To guarantee that the pruning phase always identifies a valid (cycle-free) workflow, we must be able to tell which paths lead back to set ι and which paths lead back only to themselves. We therefore additionally assign to each colored node in the supergraph G a number corresponding to the node's distance from set ι as we explore forward. Cycles are now immediately apparent in the pruning phase as a positive jump in what should be a strictly decreasing sequence of distances and thus can be avoided.

After constructing the supergraph G, we begin the exploration phase. During this phase we color the nodes reachable from set ι green and record their distances from ι. We start by coloring each label in set ι green and assigning it distance 0. We then look at the child task nodes, color those that we can green and assign them distance 1, and so forth, following the directed edges of the supergraph.

A node v cannot be colored green until all of the parent nodes that v is dependent upon (v's required inputs) are colored green. Similarly, the distance assigned to v must be greater than the distance of any parent node that v is dependent upon. If node v has only one parent then we can color v green once its parent is colored, and v's distance is simply one more than its parent's (see Figure 1.a). If node v is disjunctive, then we can color v green as soon as any one of v's parents is green and we choose v's distance to be one more than distance of the parent with the *shortest* distance (Figure 1.b). Labels are treated as disjunctive, since any one of multiple parent tasks is able to produce the output denoted by the label. If node v is conjunctive, then we must wait until all of v's parents are green before coloring v and we choose v's distance to be one more than the distance of the parent with the *longest* distance (Figure 1.c).

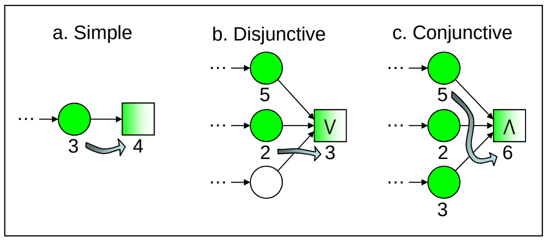

Fig. 1. Exploration Phase Coloring

As the set of green nodes grows, we continually check to see if all the nodes in set ω have been colored green. If so, we will have shown that every node in ω is reachable from set ι. If there are no further nodes we can color green and yet there are non-green nodes in ω, we will have shown that it is impossible to create a workflow composed of fragments currently known to the community that will satisfy the specification S.

Once all the nodes in ω are green, we can begin the pruning phase. During this phase, we color nodes and edges that are part of the final workflow blue. We start by coloring all the nodes in ω blue. Coloring a node blue does not change its distance number. We then work backward, selecting which edges and parent nodes to color blue, until every blue node either has a blue parent or is a member of set ι (and hence numbered with distance 0).

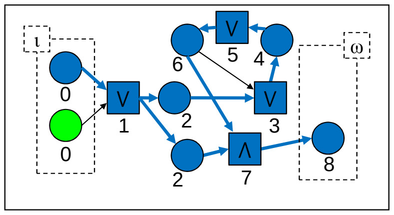

Fig. 2. Pruning Phase Coloring

For every blue node v which needs blue parents, we must select which of v's parent nodes will be used to reach v in the final workflow, and color the selected parents and connecting edges blue. If node v is conjunctive, then all of its parents are selected (see Figure 2, node 7). On the other hand, if node v is disjunctive, then we must be more careful: we must chose a parent that is reachable from ι (is colored) and that does not create a cycle (has a distance less than v's). We meet these requirements by selecting the parent with the shortest distance number, breaking ties arbitrarily (in Figure 2, at node 1 and node 3). A green selected parent is colored blue, while a blue selected parent simply remains blue; the connecting edge is colored blue in all cases. A node v chooses its required parents only once regardless of how many times v is itself identified as a required parent during the pruning phase (*e.g.*, in Figure 2, node 1). We must identify the required edges because an edge between two required nodes may be unnecessary and possibly even prohibited (in Figure 2, from node 6 to node 3). When every blue node either has blue parents or is in set ι, the algorithm is complete and the graph of blue nodes and blue edges is the desired workflow W.

Having described our algorithm in detail, we now offer a proof sketch of the correctness of our algorithm by highlighting several key invariants. First, we claim that every green node is reachable starting from ι, and all of its dependencies have a lesser distance number. A node is reachable when it is in ι, or when its dependencies are reachable. Because we start with the nodes in ι with distance 0 and we work outward one edge at a time, coloring a node v green only when v's dependencies are already green (reachable) and assigning v a distance greater than any of its dependencies, the invariant holds after every step of the algorithm.

Second, we claim that, after ω is colored blue, after each pair of steps (coloring a task and then its parent labels blue) the graph of blue nodes and blue edges is a valid workflow. At each step we choose a node v which is in the inset of the blue portion of the supergraph (as it has no blue parents). Once we color the required parents of node v blue, v is no longer a member of the inset but the required parents are now members, so v and thus v's dependents are still reachable from the inset. After each pair of steps, the sinks and sources of the graph will be labels and the graph will be a valid workflow.

Finally, we claim that the coloring of blue nodes will eventually terminate, and upon termination the graph formed by the blue nodes and edges will be a workflow satisfying specification S. From the first invariant, every node v with distance number greater than 0 must have required parents with distance strictly less than v's distance. Every time a node v in the inset is replaced with its required parents, the distance number of the nodes added to the inset is strictly less than the distance number of the node removed. Eventually the inset will consist solely of nodes with distance 0 (thus nodes in ι) and the algorithm will terminate. As the inset is a subset of ι and the outset is equal to ω, the workflow consisting of the blue nodes and edges satisfies S.

To demonstrate our algorithm in action, we offer the following simple, concrete example. A new professor at a department meeting says he has a great idea for a new class and asks how he can register a new course.

$$S(W) = (W.in \subseteq \textit{`Idea For Class'} \wedge W.out = \textit{`Class Registered'})$$

The department chair and department librarian, who are also at the meeting, contribute their knowhow in the form of workflow fragments (Figure 3). The professor collects the fragments from the community and creates the supergraph G. Next, the professor performs the exploration phase of the algorithm. Starting from set ι, the professor starts coloring nodes green and assigning distance numbers (Figure 4). Once all the nodes in set ω are green, it is time to perform the pruning phase. The professor colors all the nodes in ω blue and begins working backward, coloring the required parents and edges (Figure 5). Finally, the professor cannot color any more nodes blue. The blue nodes and edges are the resulting workflow that will allow that professor to register a new course (Figure 6). In this scenario, the branch that resulted in having a book on reserve (a potentially useful but not currently needed activity) was pruned from the workflow because it did not contribute to satisfying the specification. When a problem with a different specification is next posed to the community, this branch may in turn be part of the resulting workflow while some other branch may be pruned.

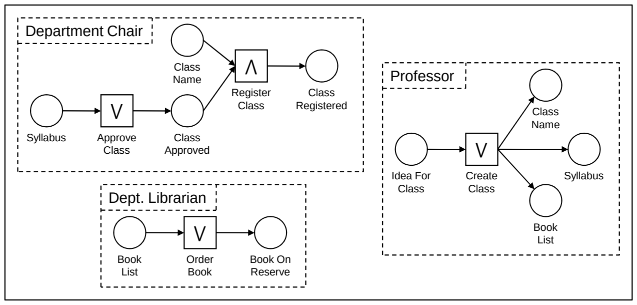

Fig. 3. Workflow Fragments from a Community

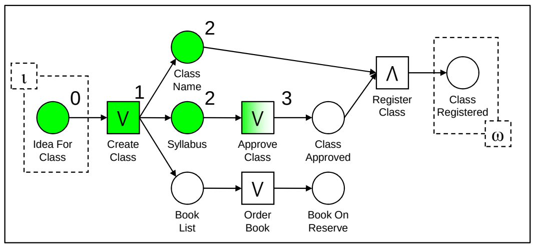

Fig. 4. Exploring the Supergraph

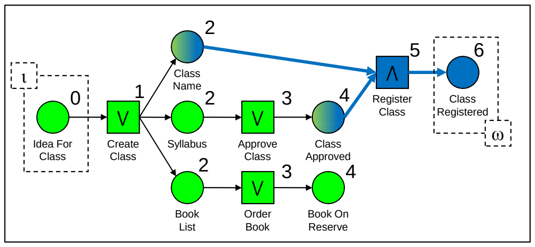

Fig. 5. Pruning the Supergraph

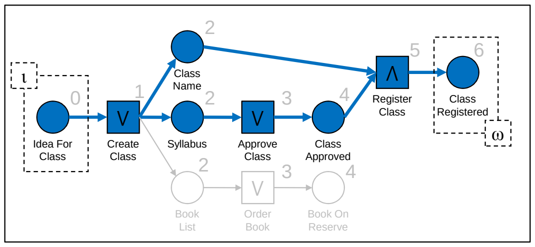

Fig. 6. The Resulting Workflow

Our basic strategy can be extended to address additional challenges. We started with the assumption that all of the workflow fragments are collected from the community and combined to form the supergraph G before the coloring process begins. Because the coloring of nodes only requires local knowledge, we can relax this requirement and build the supergraph incrementally, drawing from the community only the fragments that we need to extend the supergraph along the boundaries of the colored region.

Our algorithm provides one feasible workflow W based upon the fragment set KW and specification S, but does not guarantee that all the services required to execute W are available within the community. We can make our algorithm sensitive to service availability by first verifying that a service is available within the community before coloring a task node as reachable from the set ι.

Similarly, while a participant in the community may nominally provide a service required for a task in workflow W, scheduling conflicts may prevent that task from being allocated to any participant and thus prevent the workflow as a whole from being successfully executed. Even if a participant can be at the right place at the right time, it might not be able to receive the necessary inputs from or communicate the needed results to the appropriate members of the community in a timely fashion. To address these challenges we modify our algorithm to achieve a tighter integration with the corresponding allocation and execution strategies.

4 Distributed Workflow Allocation and Execution

In this section we summarize the steps that follow open workflow construction, *i.e.*, allocation and execution. The approach we take here is similar to prior work done for Collaboration in Ad hoc Networks (CiAN). A more in-depth discussion may be found in [8].

The participant who initiates the construction of an open workflow is considered the *coordinator*. Once the open workflow has been constructed, the coordinator is responsible for communicating with the community to allocate the tasks of the workflow to members for execution and for providing message routing during execution. For purposes of discussion, we make the simplifying assumption that the location of the coordinator is known to the community and that it can communicate with all participants.

The coordinator begins the allocation phase by computing metadata for each task (such as utility measures and topological ordering) used in allocating and executing the workflow. Next, the coordinator advertises task solicitations to the *workers* (community members who agree to participate in the execution of the workflow), who compare the task's required time, location, and service with their own capabilities and availability. If a worker finds a suitable task solicitation, and would be willing to perform the task, it submits a bid on that task to the coordinator. The bid includes information about how well the task fits with that participant's schedule, as well as the degree to which the participant is specialized for the task in question. The coordinators uses this information to select a best-suited worker to perform the task. Workers which provide fewer services are preferred over hosts with a wider array of services, because attaching a more generally capable host to a task removes a larger number of services from the pool of available services to execute all of the tasks. Workers for whom the task's timing provides a tighter fit with their schedule are preferred because their time is being used more efficiently. Workers also submit a deadline for a response from the coordinator, based on their schedule.

The coordinator selects from among the bids received, and makes a provisional task allocation to the participant whose bid best matches the selection criterion. As new bids arrive, the provisional allocation is continually re-evaluated. A final decision is made when (1) a minimum time period (either global, or based upon travel time necessary to reach the location where the task must be performed) before the task's desired execution time is reached, or (2) the deadline given by the worker who has the current provisional allocation has arrived. Thus, the coordinator waits as long as possible to assign a task to a worker in order to obtain the best possible bid, but ensures that, once someone has been found who can do a task, the task will certainly be allocated.

When a worker is allocated a task, it adds a commitment for that task to its schedule. The worker begins the process of executing the task by acquiring the necessary outputs of the preceding tasks. Because we assume a stable coordinator, all task outputs can be routed through the coordinator for reliable distribution to workers executing later tasks. Once the task has received sufficient input values to satisfy its requirements, the worker invokes the appropriate service and executes the task's activity. When the activity is completed, the worker in turn publishes the outputs of the task through the ad hoc network to the coordinator for subsequent tasks to obtain.

This basic approach to allocation and execution can be extended to be more distributed and dynamic, as was investigated in prior work [8]. The assumption of a single coordinator during allocation was relaxed and the use of multiple coordinators was investigated. The workflow can be subdivided amongst multiple coordinators according to criteria such as temporal or spatial proximity, and the coordinators can allocate their assigned tasks independently. The assumption that the coordinator is always reachable was also relaxed. Instead using centralized communication, a publish / subscribe protocol for routing messages was investigated. In this case, a participant issues subscriptions to the outputs of preceding tasks and the published outputs of those preceding tasks are returned to the host through the ad hoc network according to a scheme based on routing identifiers derived by the coordinator from a topological ordering of the tasks in the workflow. Both the number of coordinators and the routing mechanism used were evaluated in order to understand the tradeoffs.

5 System Architecture

In this section we turn to the challenge of building a real system capable of running on a physically mobile device, participating in a transient community over an ad hoc wireless network, and responding to unpredictable circumstances with the construction, allocation, and execution of an open workflow. We begin by presenting our goals, our design principles, and then our resulting architecture for an open workflow management system. We conclude with the status of our current implementation.

As open workflows can potentially execute on and take advantage of mobile devices with a great diversity of capabilities, our goal is a system that will support the coordination and participation of a wide range of devices. Further, we want to build a system robust enough and flexible enough to encourage rather than hinder the changes and innovations of future research. Consideration of these goals lead two the following two design principles.

First, the architecture should break apart the major responsibilities of the system into independent components. Keeping the components independent allows each host to provide only the components that are appropriate to the host's capabilities and to its involvement in each particular activity. While in general it is expected that all hosts are peers and any host may be capable of fulfilling any of the responsibilities in the system, this is not a requirement. For example, highly constrained devices may only be able to provide one or two services and may not be able to provide knowhow or construct a workflow. Keeping the major responsibilities separate also allows the implementation of each component to vary independently. Some component implementations can be common while other components are customized to the requirements of a particular host. For example, two hosts may share the same workflow construction component but choose to maintain their schedules in different ways.

Second, the architecture should isolate and hide the highly variable details of how communication between hosts is implemented by providing an abstract communications layer. Passing messages through an intermediary ensures that the components are decoupled and that all hosts are accessed uniformly. We do not need to handle interaction with local components differently from interaction with remote components. Using such a communications layer also allows the interfaces between components to be kept simple and expressive of the intent rather than be cluttered with details about the method of communication. The communication layer translates the request to the form most appropriate to the target host, and is free to use any transport, protocol, and caching scheme that is compatible with the interface. The interfaces between components must be written such that communication between the components has as little session specific state as possible, which encourages robustness and gives the communications layer the most leeway in how it facilitates the communication. For example, using a publish and subscribe style works well in many communication scenarios because it defines a "request" that is durable in the face of dynamic the appearance and disappearance of hosts on the network. Further, if sessionless communication methods are used, the system will never block waiting for a particular host to reply and there is no dependence on the participation of any particular host. When a communication interface can't be completely sessionless, abstract events representing host arrival and departure can be passed from the communications layer to the component to keep the component informed.

Based upon these design principles, we have identified the following major responsibilities for the open workflow management system we are developing as illustrated in Figure 7. We first observe that for a particular open workflow problem, one host acts as the coordinator while all hosts (including that coordinator) may act as participants. We therefore split the system responsibilities into two corresponding subsystems: the construction subsystem and the execution subsystem. The construction subsystem is responsible for identifying the problem to be solved, issuing queries to discover knowhow and capabilities, formulating the plan of action, and assigning work. The execution subsystem is responsible for replying to knowhow and capability queries, accepting appropriate work assignments, and actually doing the processing or communicating necessary to complete the work.

We break the construction subsystem into the Workflow Initiator, the Workflow Manager, and the Auction Manager. The Workflow Initiator is responsible for interacting with the user to define the trigger and goal conditions for the new problem. The Workflow Manager is the core component of the construction subsystem. The Workflow Manager creates and maintains a separate workspace for each open workflow, allowing it to simultaneously work on multiple isolated and independent problems. The Workflow Manager issues queries to discover knowhow and capabilities, integrates the responses into the graph, and con-

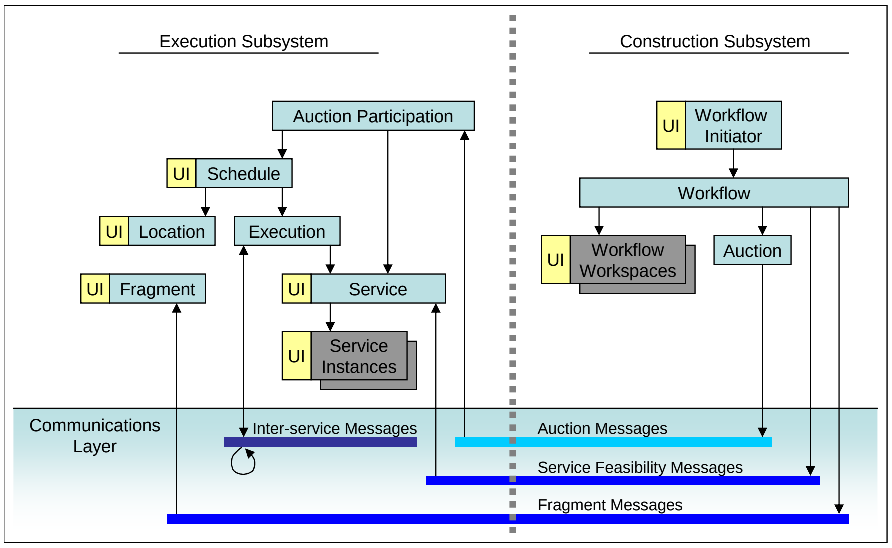

Fig. 7. System Architecture

structs the open workflow. It then delegates to the Auction Manager the job of allocating each task to a suitable host.

Separating the Workflow Initiator keeps the complexities of user interaction out of the Workflow Manager. Similarly, the auction protocol is somewhat complex and fairly orthogonal to the actual workflow construction, so we split out a separate Auction Manager. Knowhow and capability queries are fairly simple in our model, so the Workflow Manager delegates directly to the communication layer. Any caching or other enhancements to knowhow queries can be treated either as part of the communication layer, or an intermediate proxy component can be introduced.

The responsibilities of the execution subsystem can be broken down into the Fragment Manager, the Auction Participation Manager, the Schedule Manager, the Execution Manager, and the Service Manager. The Fragment Manager is responsible for maintaining a host's database of workflow fragments (its knowhow) and responding to knowhow queries during workflow construction. The Auction Participation Manager again encapsulates the interactional complexity and state tracking needed for the host to bid in task auctions during the allocation phase. The Schedule Manager is the keystone component of the execution subsystem. It manages the host's availability by tracking the host's location, schedule, and scheduling preferences. It maintains a database of all commitments, primarily consisting of scheduled service invocations and their associated location and travel time details, which is the key data structure for both allocation and execution of an open workflow. The Schedule Manager also generates time based and location based notifications (such as notifying a user that it is time to start moving to a new location — important for helping the human participants in the open workflow system keep their commitments). The Execution Manager monitors the input message and time conditions required for each scheduled service invocation during the execution phase. Once the necessary conditions are met, it triggers service execution, and publishes any output messages. Finally, the Service Manager maintains the list of services exposed by this host, responding to capability queries from the Workflow Manager. It also provides a uniform service invocation interface to the Execution Manager by handling parameter marshaling and any other mechanics required to actually invoke a local service during the execution phase.

The Fragment Manager is orthogonal to the rest of the execution subsystem. The Auction Participation Manager is separated from the Schedule Manager because it again encapsulates a fairly complex protocol. The managing of local services is a well defined zone of responsibility appropriate for a single component. The Execution Manager exists to pull the inter-service messaging and condition monitoring responsibilities out of the Schedule Manager. The Schedule Manager still has multiple responsibilities assigned to it, but these responsibilities are thus more tightly interrelated. Simple hosts may find it convenient to implement these responsibilities in a single component. For example, a minimal host such as a remote sensing beacon might have limited to effectively nonexistent mobility and very few service options. Such a host could have a scheduler as simple as a bit array indicating which hours of the day the sensor should take and report a measurement. A full featured host with a database of locations and travel time, manually specified commitments, and manual intervention during scheduling conflicts will probably choose to break the Schedule Manager down into further subcomponents.

Each component we have described in the construction and execution subsystems makes a distinct and important contribution in moving an open workflow from the initial problem definition to an executed solution. The Schedule Manager handles spatial-temporal information and constraints. The Fragment Manager and Service Manager provide discovery of the knowhow and capabilities available in the current environment. The auction components provide adaptive allocation given these complex constraints. The communication layer abstracts away the greatly varying reachability of hosts. The remaining few components (the Workflow Initiator, Workflow Manager, and Execution Manager) provide decoupling to make the architecture more robust and isolation to provide scalability. The design of our architecture is deeply motivated by the challenges posed by open workflows.

Our architecture permits multiple open workflows to be constructed and executed concurrently within the same spatiotemporal context. The Workflow Manager must maintain a separate workspace containing construction state information for each open workflow. The remaining components (such as the Auction Manager, Fragment Manager, Schedule Manager, etc.) act at task granularity and thus handle two task-based requests from two separate workflows no differently than they handle two task-based requests from the same workflow. While multiple workflows will necessarily compete for utilization of the same resources (in the form of hosts, their capabilities, and other resources present in the environment), there is no impedance at an architectural level to constructing and executing multiple open workflows.

Based upon this architecture, we have designed and implemented an initial open workflow management system focused primarily on testing and simulation. Our system is being evaluated and refined, but based informally on our experience so far the architecture appears sound. As our next step, we are working to replace our simulation communications layer with a real ad hoc wireless network communication layer and evaluate the otherwise identical system on real mobile devices.

6 Related Work

In this paper, we have been focused on overcoming the challenges of bringing workflows to transient communities connected by mobile ad hoc networks and faced with unpredictable situations. Standard workflow management systems, such as ActiveBPEL [10], Oracle Workflow Engine [11], JBoss [12], and BizTalk [13], are designed to work in fully wired environments, such as corporate LANs or across the Internet. Reliance on centralized control and reliable communication mean such solutions cannot successfully operate under the constraints of dynamic mobile environments.

Several workflow systems have been developed which extend the realms in which workflows may operate. The work on federating separate execution engines running independent workflows by Omicini, *et al.*, [14] removes the requirement for centralized control. Chafle, *et al.*, [15], investigate decentralized orchestration of a single workflow by partitioning the workflow at build time and using message passing at run time. Both approaches still assume reliable communication and a fixed group of participants. MoCA [16] uses proxies for distributed control and has some design features that support mobile environments while Exotica/FDMC [17] describes a scheme to handle disconnected mobile hosts. In AWA/PDA [18], the authors adopt a mobile agent based approach based on the GRASSHOPPER agent system. WORKPAD [19] is designed to meet the challenges of collaboration in a peer-to-peer MANET involving multiple human users, however WORKPAD retains the requirement that at least one member of the MANET be connected with a central coordinating entity that orchestrates the workflow and shoulders any heavy computational loads. Sliver [7] brings a full BPEL execution engine to a single cell phone, however that phone still acts as the sole coordinator. Finally, CiAN [8] presents a workflow management system which eliminates the need for a central arbiter by distributing not only service execution but also the task allocation problem across multiple hosts.

While our system builds upon CiAN's model of distributed workflow allocation and execution, all these assume that a thoughtfully designed and fully specified workflow already exists. Open workflows are designed for settings where the availability of resources and the range of responses demanded by changing circumstances cannot be anticipated. The workflow to be executed must be generated on the fly to match the present situation.

The automatic composition of services has been explored using a variety of AI planing engines, including Golog [20], Workflow Prolog [21], and PDDL [22]. A review of further automated service composition methods may be found in [23]. Ponnekanti and Fox create workflows by rule-based chaining in SWORD [6], and point out that the resulting workflows may not produce the desired results if the preconditons and postconditions of each task are not sufficiently specified. Fantechi and Najm [24] present an approach for ensuring correct service composition by using a more detailed formal specification of the service behavior. While the initial open workflow construction algorithm we present is a simplification of the powerful techniques presented in these papers, it also addresses a new problem specific in the mobile ad hoc environment. All these systems assume that the knowledge base from which to build the workflow already exists. We have built upon their work by showing how to construct both the knowledge base and the derived workflow on the fly based on the knowhow and capabilities available within the community.

7 Conclusions

In this paper we introduced the open workflow paradigm and presented a first algorithm for constructing open workflows in ad hoc wireless mobile environments. An architecture supporting open workflow creation, allocation and execution has been proposed and a first instantiation of this architecture has been built.

The approach is novel and makes possible the development of new classes of applications that are designed to exploit community knowledge in solving real world problems that arise unexpectedly and can be addressed only through the coordinated exploitation of capabilities distributed among the members of the community. The open workflow paradigm presents our research community with significant new challenges in the technical, system, and application domains. The work presented in this paper is only the first step towards characterizing and addressing these concerns.

We have demonstrated the feasibility of building an open workflow framework. While high impact practical solutions will require both further refinements and the shaping of the paradigm to the needs of each particular application domain, the more abstract formulation of the problem presented here opens interesting opportunities for formal studies complementing the system design and evaluation efforts.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
2. OASIS: Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf` (April 2007)

3. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet another workflow language. Information Systems **30**(4) (2005) 245–275
4. Swenson, K.D., Pradhan, S., Gilger, M.D.: WfXML 2.0: XML based protocol for run-time integration of process engines. `http://www.wfmc.org/standards/docs/WfXML20-200410c.pdf` (October 2004)
5. Thatte, S.: XLANG: Web services for business process design. `http://www.gotdotnet.com/team/xml\_wsspecs/xlang-c/default.htm` (2001)
6. Ponnekanti, S.R., Fox, A.: SWORD: A developer toolkit for web service composition. In: Proceedings of the 11th World Wide Web Conference, Honolulu, Hawaii, USA (May 2002)
7. Hackmann, G., et al.: Sliver: A BPEL workflow process execution engine for mobile devices. In: LNCS. Volume 4294. (2006) 503–508
8. Sen, R., Roman, G.C., Gill, C.D.: CiAN: A workflow engine for MANETs. [25] 280–295
9. Handorean, R., Gill, C.D., Roman, G.C.: Accommodating transient connectivity in ad hoc and mobile settings. In Ferscha, A., Mattern, F., eds.: Pervasive. Volume 3001 of Lecture Notes in Computer Science., Springer (2004) 305–322
10. Active-Endpoints: ActiveBPEL engine. `http://www.active-endpoints.com/active-bpel-engine-overview.htm`
11. Oracle Inc.: Oracle workflow. `http://www.oracle.com/technology/products/integration/workflow/workflow_fov.html`
12. JBoss Labs: JBoss application server. `http://www.jboss.com/docs/index`
13. Microsoft Corp.: The BizTalk server. `http://www.microsoft.com/biztalk/`
14. Omicini, A., Ricci, A., Zaghini, N.: Distributed workflow upon linkable coordination artifacts. In Ciancarini, P., Wiklicky, H., eds.: COORDINATION. Volume 4038 of Lecture Notes in Computer Science., Springer (2006) 228–246
15. Chafle, G., Chandra, S., Mann, V., Nanda, M.G.: Decentralized orchestration of composite web services. In: Proc. of the 13th Intl. WWW Conference. (2004) 134–143
16. Sacramento, V., et al.: An architecture supporting the development of collaborative applications for mobile users. In: Proc. of WETICE '04. (2004) 109–114
17. Alonso, G., Gunthor, R., Kamath, M., Agrawal, D., Abbadi, A.E., Mohan, C.: Exotica/FDMC: A workflow management system for mobile and disconnected clients. Parallel and Distributed Databases **4**(3) (1996)
18. Stormer, H., Knorr, K.: PDA- and agent-based execution of workflow tasks. In: Proceedings of Informatik 2001. (2001) 968–973
19. Mecella, M., Angelaccio, M., Krek, A., Catarci, T., Buttarazzi, B., Dustdar, S.: WORKPAD: an adaptive peer-to-peer software infrastructure for supporting collaborative work of human operators in emergency/disaster scenarios. Collaborative Technologies and Systems, International Symposium on **0** (2006) 173–180
20. McIlraith, S., Son, T.C.: Adapting golog for composition of semantic web services. In: Proceedings of the 8th International Conference on Knowledge Representation and Reasoning(KR2002). (2002) 482–493
21. Gregory, S., Paschali, M.: A prolog-based language for workflow programming. In Murphy, A.L., Vitek, J., eds.: COORDINATION. Volume 4467 of Lecture Notes in Computer Science., Springer (2007) 56–75
22. McDermott, D.: Estimated-regression planning for interactions with web services. In: Proceedings of the 6th International Conference on AI Planning and Scheduling, AAAI Press (2002) 204–211

23. Rao, J., Su, X.: A survey of automated web service composition methods. In: In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, Springer-Verlag (2004) 43–54
24. Fantechi, A., Najm, E.: Session types for orchestration charts. [25] 117–134
25. Lea, D., Zavattaro, G., eds.: Coordination Models and Languages, 10th International Conference, COORDINATION 2008, Oslo, Norway, June 4-6, 2008. Proceedings. In Lea, D., Zavattaro, G., eds.: COORDINATION. Volume 5052 of Lecture Notes in Computer Science., Springer (2008)