Achieving Coordination through Dynamic Construction of Open Workflows

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

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

Abstract. Workflow middleware executes tasks orchestrated by rules defined in a carefully handcrafted static graph. Workflow management systems have proved effective for service-oriented business automation in stable, wired infrastructures. We introduce a radically new paradigm for workflow construction and execution called open workflow to support goal-directed coordination among physically mobile people and devices that form a transient community over an ad hoc wireless network. The quintessential feature of the open workflow paradigm is dynamic construction of custom, context-specific workflows in response to unpredictable and evolving circumstances by exploiting the knowledge and services available within a given spatiotemporal context. This paper introduces the open workflow approach, surveys open research challenges in this promising new field, and presents algorithmic, architectural, and evaluation results for the first practical realization of an open workflow management system.

1 Introduction

With the development of small, powerful wireless devices, computing must embrace the frequent, transient, ad hoc interactions of mobile environments. As computing and communication become more and more integrated into the fabric of our society, new kinds of enterprises and new forms of social interactions will continue to emerge. We ask the fundamental question: how can ad hoc communities of people (and their personal devices) coordinate to solve problems? Application domains that motivate or even require this form of interaction 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 training. She initiates the response, but access to the spill is made difficult by a support structure whose dismantling requires special intervention which only the chief engineer can manage. The

J.M. Bacon and B.F. Cooper (Eds.): Middleware 2009, LNCS 5896, pp. 268–287, 2009.

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 knowledge and resources.

Current workflow middleware allows people to initiate complex goal-oriented activities that leverage services made available by a wide range of service-oriented portals. In the typical scenario, a user employs a web browser to make a request to a workflow engine responsible for executing a predefined workflow that can satisfy the specific user need, e.g., to print photos, reserve tickets, or make a bid in an online auction. The workflow is 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 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's need and the workflow required to satisfy it, between the task specifications and the services that implement them, and between the workflow engine that invokes a service and the service provider that executes it.

Despite workflow middleware being well established, efforts toward using it in ad hoc wireless environments are relatively new. Our previous research in this area includes the development of workflow execution engines targeted to small portable devices [1], and techniques for executing workflows in mobile wireless networks [2]. These studies reveal the need for a major reevaluation of the way one thinks about workflow middleware: hosts may move, service availability may depend 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.

Starting with this premise, we pose the question of how workflow middleware might be reshaped for use *in the absence of any wired connectivity*. In this paper, we explore whether workflow middleware 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 activity whose underlying infrastructure is a mobile ad hoc wireless network. We assume a set of participants (people and the host devices they carry) who share a sense of purpose and who can move about and interact with each other and with the real world. The participants form a transient community that evolves over time. In our approach, one of the members of a community identifies a need for action, which then results in the dynamic construction of a workflow to satisfy the need and the execution of that workflow in a distributed and cooperative manner. The defining feature of the open workflow paradigm is the workflow construction process: workflow fragments encoding individual knowledge distributed across the set of participants are assembled into a custom workflow both automatically and contextually. 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 time and to delivering results to any dependent participants. The latter highlights another feature of the open workflow paradigm, its sensitivity to the time and location considerations necessary when performing activities in the real world.

Exploring the challenges of building a workflow on the fly from available contextual knowledge, i.e., the open workflow paradigm, and building a platform for further experimentation with that approach define the core technical contribution of this paper. We present a formalism for describing open workflow construction in Section 2. Section 3 explains our algorithms for the collaborative construction, allocation, and execution of open workflows. In Section 4, we present our open workflow management system and discuss its architecture. In Section 5, we evaluate its performance and discuss directions for future work. Section 6 highlights related research and contrasts it with this work. We provide conclusions in Section 7.

2 Problem Definition

2.1 Motivating Example

To highlight the possibilities and advantages of the open workflow paradigm, consider how a corporate catering facility might use open workflow to organize meals for a meeting of corporate executives. Suppose an executive assistant calls the manager at the catering office and requests breakfast and lunch for the upcoming meeting. The manager adds the request to the open workflow system on her mobile device to schedule the activities necessary to prepare the meals for the meeting. The open workflow engine begins by collecting knowledge contained on other mobile devices owned by the employees in the catering office, which include a master chef, kitchen staff, wait staff, and other personnel. For example, the master chef's PDA contains a workflow fragment consisting of tasks and conditions that describe how to serve omelets for breakfast. Figure 1 shows the collection of workflow fragments obtained from the office community.

Using the available knowledge, the open workflow engine searches for a subgraph that meets the conditions and requirements given by the manager. Assuming breakfast and lunch ingredients are available, we see that setting up an omelet bar and cooking omelets will result in breakfast being served, and preparing a soup and salad and setting them out as a buffet will result in lunch being served. Thus, this sub-graph constitutes a workflow that meets the requirements. The open workflow engine then searches for participants that are able to perform the activities indicated by the sub-graph. The system schedules the kitchen staff to set out the ingredients for breakfast and an appointment is added to the master chef's PDA to cook the omelets. Similarly, the kitchen staff must later cook lunch and set out the buffet. The manager's work is complete, and the members of the staff go about their scheduled activities.

Clearly, changes in the requirements will affect the generated workflow. For example, if lunch was not requested, then no lunch activities will be included

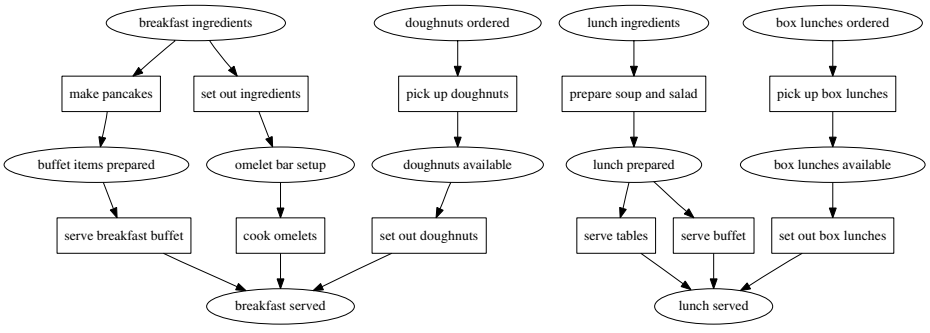


Fig. 1. Available knowledge in a corporate catering facility

in the final workflow. Consider a scenario where the master chef is out of the office. The workflow fragment concerning the preparation of omelets will never be collected and considered by the workflow engine. Consequently, one of the two alternatives (coffee and doughnuts or a breakfast buffet) will be chosen instead. A similar scenario where the wait staff are absent demonstrates how changes in the set of available capabilities can affect the construction of the workflow. The master chef knows that lunch can either be served with buffet service or with table service but the open workflow engine must select buffet service since no one in the available community is capable of serving tables.

Consider the difficulties of using a traditional static workflow to manage the catering facility. To be sensitive to the variety of catering requests and the individual capabilities and dynamic availability of the employees, the workflow would contain a large number of conditional branches which must be carefully crafted and assiduously maintained. Such a static workflow cannot respond rapidly to new resources or changes in the environment. Sensitivity to context, in the form of knowledge, capabilities, and availability, is the driving force behind the creation of our open workflow system.

2.2 Formalization

A *workflow* is defined as a collection of interlinked abstract *tasks*. A task represents a single abstract behavior or accomplishment without completely specifying how it must be performed. A *service* is a concrete implementation of a task and may involve a computation by the device, an activity performed by the user, or some combination of the two. Execution of a task thus consists of the invocation of a service satisfying the respective task specification. Within a workflow, different tasks may be performed in sequence or in parallel by one actor or by multiple actors. Each task has *preconditions* that must be met before the task can be performed, and *postconditions* that describe the results of performing the task. Abstractly, we can enable the performance of a given task by performing one or more preceding tasks whose postconditions taken together ensure the preconditions necessary for the given task. The order, timing, and executors of the

preceding tasks are unconstrained so long as the given task's preconditions hold when it is to be performed.

We assume that each input (precondition) and output (postcondition) of a task is represented by a *label*, where each label has a distinct meaning. We also assume that a task is either *conjunctive*, requiring all of its inputs, or *disjunctive*, requiring only one of its inputs, and that a task produces all of its outputs. Tasks are joined by matching the labels on inputs and outputs exactly. Labels and tasks within a workflow thus may be considered nodes in a bipartite directed acyclic graph. We assume that each node has a semantic identifier; nodes with the same identifier are equivalent.

A workflow has the additional constraints that (1) all sources (nodes without any incoming edges) and all sinks (nodes without any outgoing edges) are labels, (2) a label can have at most one incoming edge, and (3) there are no duplicate nodes in the graph. This definition allows us to *compose* two workflows by merging (a) identical sinks from one workflow with the corresponding sources from the other workflow and (b) identical sources in both workflows. Two workflows are *composable* if and only if matching sinks and sources yields a valid workflow. For instance, a workflow W_1 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 $\{f, g, h\}$. Workflow *fragments* are merely small workflows (possibly even a single task) that are intended to be composed into larger workflows at a later time. In the example graph in Figure 1, the boxes are tasks and the ovals are labels. The graph represents the available knowledge of the catering facility but is *not* a valid workflow because some labels have multiple incoming edges.

A workflow is constructed in response to an expressed need. In general, this need is stated in terms of a *specification* S: a *predicate* that indicates whether or not a workflow is *satisfactory*. The *inset* and the *outset* of a workflow are its sources and sinks respectively. We assume S is of the form

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

A workflow W with inset $W.in$ and outset $W.out$ then satisfies a specification S if and only if $S(W.in, W.out)$ is true.

Composing workflow fragments may produce a workflow that cannot satisfy a specification S only due to the existence of extra sinks or sources. We can prune a workflow to remove unnecessary data flows, subject to the following constraints which ensure the result remains a valid 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 any task outputs that are sinks are also pruned.

Once a problem has been identified and a specification given, the *knowhow* (in the form of workflow fragments) and *capabilities* (in the form of services) of the local community are synthesized to form a plan by constructing a workflow. The construction problem is defined as follows. Given a workflow specification

S and a set of workflow fragments K, find a set of workflow fragments in K which may be composed (subject to pruning) into a workflow W that satisfies S — we say that W is *feasible* given S and K. 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 new need, context change, or other event, and that the set K represents the combined knowledge of the community as a whole. K is distributed and dynamic. As participants move around in space, the knowledge available to the community changes with its membership and their experiences. For the same specifications, different communities may respond differently or may be unable to construct an appropriate workflow.

As the plan is formed, tasks must be *allocated* to participants who will eventually *execute* corresponding services. The *availability* of services and resources within the community determines to whom tasks are allocated. Service availability is determined by whether any participant can commit to providing a service: that is, (1) whether the participant is *capable* of performing the service, (2) whether the participant has time available, (3) whether the participant can travel to the necessary location to perform the service, (4) whether the participant can gather the necessary inputs and distribute any outputs in a timely manner, and (5) whether the participant is willing (according to their preferences) to perform the service. If the community is stable and all participants are mutually reachable, it is easy to guarantee that the participants supporting the execution of tasks that depend upon each other are able to communicate the needed results in a timely fashion. More sophisticated routing techniques and analysis [3] may be needed if the movement of participants results in temporary disconnections. Once a participant has made a commitment, it is responsible for ensuring the service is executed as agreed. A participant is thus free to move about and requires no further communication with the community except possibly for previously agreed upon meetings to gather inputs or distribute outputs. As individual participants execute their assigned services from the dynamically constructed workflow, the community as a whole thus performs the activities necessary to satisfy the specification and achieve the original goal.

3 Collaborative Construction, Allocation, and Execution

3.1 Construction

We begin this section by introducing a construction algorithm for open workflows. We assume a participant has identified a need for action and generated a specification S 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 participant is in contact with the other members of a community and can collect from each

a set of workflow fragments. For the purposes of illustration, we start with the simplifying assumption that the participant initially collects all the fragments in the community to create the set K. Using the gathered information, the participant runs our algorithm to find a feasible workflow — a workflow composed of fragments from K (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 K to satisfy S. While our algorithm chooses arbitrarily among equivalent options, any heuristic may be incorporated to direct the search toward more favorable solutions.

Our algorithm is based on graph traversal and graph coloring, and takes its inspiration from spanning tree algorithms and routing algorithms such as AODV [4]. Our strategy is to combine all workflow fragments from K into one large graph, henceforth called the workflow *supergraph* G. The supergraph represents a unified view of all possible actions represented in the set K, however it is not necessarily a valid workflow since it may have cycles, outputs produced by multiple tasks, unavailable inputs, or undesired 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 ω, we 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 that is composed only of fragments in K that have been pruned of unneeded outputs and paths.

With this general strategy in mind, we present the full pseudo-code in Algorithm 1. For purposes of the algorithm, we annotate every node and edge in G with a *color* (initially *uncolored*) and every node with a *distance* (initially ∞) from a source on the graph. Nodes are marked *green* for reachability during the exploration phase and *blue* for workflow membership during the pruning phase; *purple* identifies nodes on the boundary of the blue region. Label nodes are considered disjunctive. The algorithm selects nodes nondeterministically; any node may be processed next so long as it matches the guard condition.

We 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 prerequisites have a smaller distance. A node is reachable when it is in ι, or when its prerequisites are reachable. The invariant holds after every step of the algorithm because we start with the nodes in ι with distance 0 and we work outward one edge at a time, coloring a node n green only when n's

Algorithm 1. Workflow Construction (given ι, ω, and K)

— *Construct Supergraph* —
$G \leftarrow \emptyset$
for all fragments $F \in K$ **do**
  for all nodes $n \in F$ **do** **if** $n \notin G$ **then** $G \leftarrow G \cup \{n\}$ **end if** **end for**
  for all edges $e \in F$ **do** **if** $e \notin G$ **then** $G \leftarrow G \cup \{e\}$ **end if** **end for**
end for

— *Exploration Phase* —
Track the set of *greenNodes* (initially empty).
for all $n \in \iota$ **do** (*n.color*, *n.distance*) $\leftarrow$ (*green*, 0) **end for**
until $\omega \subseteq$ *greenNodes* $\vee$ none of the following cases apply, **for** some $n \in G$ **do**
  if n is disjunctive $\wedge$ any of n's parents are *green* **then**
    $d \leftarrow \min\{p \in n\text{'s parents} \vee p.color = green \,|\, p.distance\}$
    if $(n.color = uncolored \vee (n.color = green \wedge n.distance > d+1))$ **then**
      (*n.color*, *n.distance*) $\leftarrow$ (*green*, $d+1$)
    end if
  else if n is conjunctive $\wedge$ all of n's parents are *green* **then**
    $d \leftarrow \max\{p \in n\text{'s parents} \vee p.color = green \,|\, p.distance\}$
    if $(n.color = uncolored \vee (n.color = green \wedge n.distance > d+1))$ **then**
      (*n.color*, *n.distance*) $\leftarrow$ (*green*, $d+1$)
    end if
  end if
end until
if $\neg(\omega \subseteq$ *greenNodes*$)$ **then** there is no solution — exit.

— *Pruning Phase* —
Track the set of *purpleNodes* (initially empty).
for all $n \in \omega$ **do** *n.color* $\leftarrow$ *purple* **end for**
until *purpleNodes* $= \emptyset$ **for** some $n \in$ *purpleNodes* **do**
  if *n.distance* $= 0$ **then**
    requiredParents $\leftarrow \emptyset$
  else if n is disjunctive **then**
    requiredParents $\leftarrow$ {the parent of n with minimum *distance*}
  else if n is conjunctive **then**
    requiredParents $\leftarrow$ n's parents
  end if
  for all $p \in$ *requiredParents* **do**
    edge(p, n).*color* $\leftarrow$ *blue*
    if *p.color* $=$ *green* **then** *p.color* $\leftarrow$ *purple* **end if**
  end for
  n.color $\leftarrow$ *blue*
end until
The set of nodes and edges colored *blue* is the constructed workflow.

prerequisites are already green (reachable) and assigning n a distance greater than any of its prerequisites.

Second, once ω is colored blue, we claim that after every even number of iterations, the graph of blue nodes and blue edges is a valid workflow. At each step we choose a node n which is in the inset of the blue portion of the supergraph as it has no blue parents. Once we color the prerequisites of n blue, n is no longer a member of the inset but the prerequisite nodes are now members, so n and thus n's dependents are still reachable from the inset. On an odd iteration we color a task, and on the even iteration we color its prerequisite labels. Thus, 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 n with distance greater than 0 must have prerequisites with distance strictly less than n's distance. Every time a node n in the inset is replaced with its prerequisites, the distance of the nodes added to the inset is strictly less than the distance 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.

While there are many ways to maintain a community and share knowledge within that community, we chose an approach that places few restrictions on the members. We define a community as the participants who are within communication range of each other and announce their willingness to participate; consequently, the community is dynamic as members join and leave at will.

We observe that the coloring process requires only local knowledge. Thus, we relax the assumption that all of the workflow fragments are collected from the community before the coloring process begins. In our implementation, the member constructing the workflow builds the set of workflow fragments K and thus the supergraph G incrementally by querying other members of the community for workflow fragments that can be used to extend G. Members joining after the algorithm has started can still contribute knowledge, and the departure of a member does not affect the knowledge already collected in the supergraph.

3.2 Allocation and Execution

After a workflow is constructed, it must be allocated to participants in the community. The approach we take here is an auction algorithm similar to prior work done for Collaboration in Ad hoc Networks (CiAN). A more in-depth discussion may be found in [2].

The participant who constructs the workflow assumes the role of *auction manager*. The auction manager begins the allocation phase by computing metadata for each task used in allocating and executing the workflow. Next, the auction manager solicits bids for each task in the workflow from all of the participants in the community. The participants compare the task's required time, location, and

service with their own capabilities and availability. If a participant can commit to performing a task, it submits a firm bid on that task to the auction manager. The bid includes ranking information such as the degree to which the participant is specialized for the task in question. The auction manager uses this information to select a best-suited participant to perform the task. A participant which provides fewer services is preferred over a participant with a wider array of services, because scheduling the more capable participant removes a larger number of services from the community's resource pool. Participants also submit a deadline for a response from the auction manager based on their schedule.

The auction manager selects the bid that best matches the selection criterion and makes a tentative task allocation to that participant. As new bids arrive, the tentative allocation is continually re-evaluated. A final decision is made when the deadline given by the participant who has the current tentative allocation has arrived. The auction manager waits as long as possible to assign a task to a participant in order to obtain the best possible bid, but once some participant has been found who can do a task, the task is guaranteed to be allocated. As bids are firm, a participant cannot cancel a bid, but they can update the deadline for a bid and force the auction manager to make a decision.

When a participant is allocated a task, it adds a commitment to its schedule that contains all the necessary information to execute the appropriate service as directed by the auction manger. The participant is free to roam, but is responsible for meeting its commitments. Thus the execution phase of an open workflow proceeds in a fully decentralized, distributed manner. To meet a commitment, the participant must (1) acquire the required inputs for the service from the executor of the preceding tasks, (2) be at the required location for executing the service, and (3) execute the service at the required time. The participant monitors these conditions and, based upon their knowledge of their location and the travel times involved, travels and communicates as necessary to meet the conditions and successfully execute the service. Once the service has been executed, the participant's final responsibility is to communicate the service's outputs to any other participants that require them.

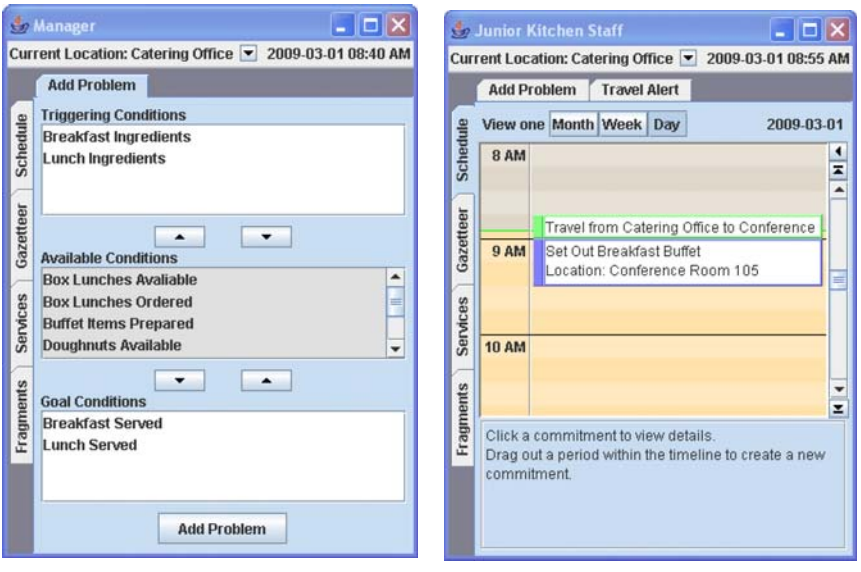
4 System Architecture

4.1 An Open Workflow Management System

We have designed and implemented a complete open workflow management system in Java. Our approach offers an intuitive calendar-like interface, behind which integrated goal specification, communication, and service invocation features combine to enable construction and execution of sophisticated open workflows. Source code and executables for the application are available at our web site [5].

The basic steps in deploying an application using our open workflow management system are (1) installing the program on the users' devices, (2) adding knowhow in the form of workflow fragments, and (3) adding service descriptions. In our implementation, we use XML configuration files to provide the task and

service definitions for each device. Once this initial configuration has been completed, any participant can use their device to create a problem specification. In response, the system will automatically construct, allocate, and (by prompting the users) execute an appropriate workflow.



(a) Add Problem Tab (b) Schedule Tab

Fig. 2. Application Screenshots

Figure 2 shows two screenshots from community members participating in an open workflow. The tabs on the left are for reviewing static knowledge. On the top are tabs for dynamic activities and alerts. Figure 2(a) shows the form that allows the user to create a problem specification by entering information about the triggering conditions and goal. In Figure 2(b), the Schedule tab allows the user to view their schedule of commitments. The necessary travel time is also blocked out in the schedule, and the system has added an alert tab to notify the user that they must soon begin traveling to meet their scheduled commitment. The system supports services that require user action by presenting a form for data entry or just a button to click when the task is complete. The remaining tabs allow the user to configure the list of workflow fragments (knowhow), the list of local services (capabilities), and other system settings.

4.2 Goals, Design Principles, and Architecture

Our goal is a system that will support the coordination and participation of devices with diverse capabilities. Further, we want to build a system robust

enough and flexible enough to encourage rather than hinder innovations from future research. Consideration of these goals led us to the following two design principles. First, the architecture should break apart the major responsibilities of the system into independent components, allowing each host to provide only the components that are appropriate to the host's physical capabilities. Second, the architecture should isolate and hide the highly variable details of the transports, protocols, and caching schemes used during communication by providing an abstract communications layer. Furthermore, passing messages through an intermediary ensures that local and remote components are accessed uniformly.

Based upon these design principles, we identified the following major responsibilities for our open workflow management system, as illustrated in Figure 3. We first observe that for a particular open workflow problem, one host acts as the initiator while all hosts (including the initiator) 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.

Construction Subsystem. The Workflow Initiator is responsible for interacting with the user to define the trigger conditions and goal 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

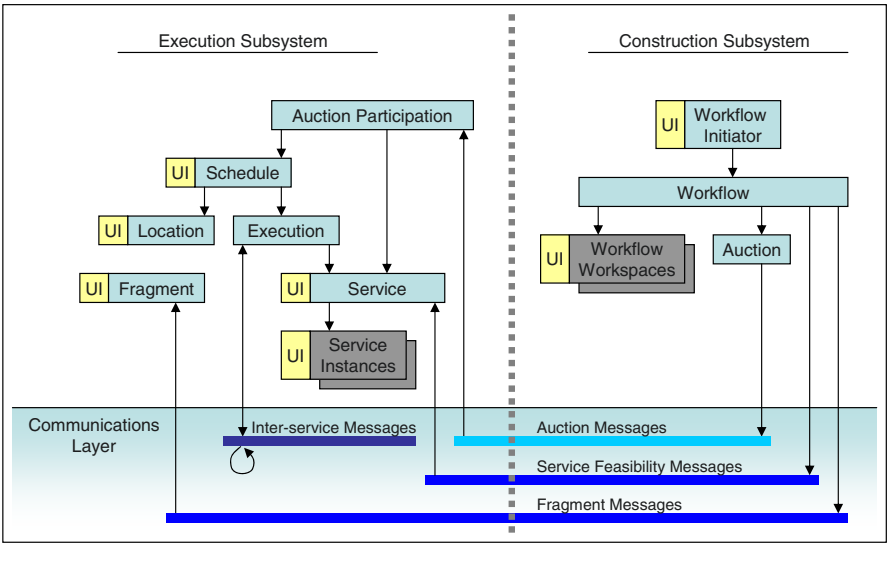


Fig. 3. System Architecture

workflow, allowing it to work simultaneously on multiple isolated and independent problems. The Workflow Manager issues queries to discover knowhow and capabilities, integrates the responses into the graph, and constructs the open workflow. It then delegates to the Auction Manager the job of allocating each task to a suitable host.

Execution Subsystem. The Fragment Manager is responsible for maintaining a host's database of workflow fragments and responding to knowhow queries during workflow construction. The Auction Participation Manager encapsulates the complex interactions 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 Execution Manager monitors the input and temporal conditions required for each scheduled service invocation during the execution phase. Once an invocation's 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 and responds 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.

Our architecture permits multiple open workflows to be constructed and executed concurrently within the same community and even within the same host. The Workflow Manager maintains a separate workspace containing construction state information for each 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 at once.

5 Evaluation

We use a combination of simulation and empirical evaluation to test our system and demonstrate the viability of the open workflow paradigm. We focus on characterizing the performance of the system in terms of three variables that have the greatest impact on the scalability of our architecture: the number of participants in the community, the number of tasks known to the entire community, and the difficulty of the problem being solved which we characterize by the size of the resulting workflow.

Our experimental set up is as follows. Given the number of hosts, the global number of tasks, and the length of the workflow as parameters for an experiment, we configure the hosts, establish connectivity within the community, and then measure the time taken from when the specification is given to the initiating host to the time when all tasks of the resulting workflow have been successfully allocated to some host.

To configure the hosts, we first construct a workflow supergraph of the chosen size by creating the desired number of nodes and then repeatedly adding edges between disconnected nodes until the graph is strongly connected. From this single supergraph we can then draw a large number of guaranteed-satisfiable specifications by randomly picking any triggering conditions and goal. We use only disjunctive task nodes in order to maintain the guarantee of satisfiability during our automated evaluations. Given a supergraph and a chosen number of hosts, we finish setting up the scenario by distributing the tasks randomly and evenly amongst the hosts, and *independently* distributing corresponding services randomly and evenly amongst the hosts. Each of the n hosts has only $\frac{1}{n}$th of the entire supergraph, so the hosts must cooperate to solve the posed problem. For each test run, the test driver randomly choses a path of the desired length through the supergraph, and the initial and final label nodes of the path are used as the specification for that test run. In all of the figures below, the results for each path length are the average of one thousand runs.

For the simulations, all the hosts were run within in a single JVM and communicate solely through a simulated network. The simulations were run on a Windows XP workstation with a 2.8 GHz Intel Xeon processor and 2.75 GB of memory, running the Java 1.6.0_11 HotSpot Client VM.

In Figure 4, we show the average time for each path length from a supergraph with 100 task nodes as the number of participating hosts varies from 2 to 15. The average time grows roughly linearly with the number of hosts as the initiating host communicates pairwise with every member of the community during the construction and allocation phases. We note that even if we were to broadcast requests rather than using pairwise communication, the processing of responses by the initiating host would still require time linear in the number of hosts in the community.

In Figure 5, we show the average time for each path length for 2 participating hosts as the number of task nodes in the supergraph varies from 25 to 500. The rate of increase grows with the number of task nodes because the Workflow Manager encounters more nodes during its search through the densely connected supergraph as the number of tasks increases. The longest path through the graph also increases as the size of the graph increases, which explains the absence of timings for path lengths greater than 10 in the small 25 task supergraph.

After the simulations, we performed empirical evaluation of our application using four laptops connected by an ad hoc wireless network using 802.llg (54Mbit/s). The first host (which was the initiating host during these tests) was a MacBook Pro running OS X 10.5.5 with a 2.16 GHz Intel Core Duo processor and 1 GB of 667 MHz DDR2 memory. The second host was a MacBook Pro

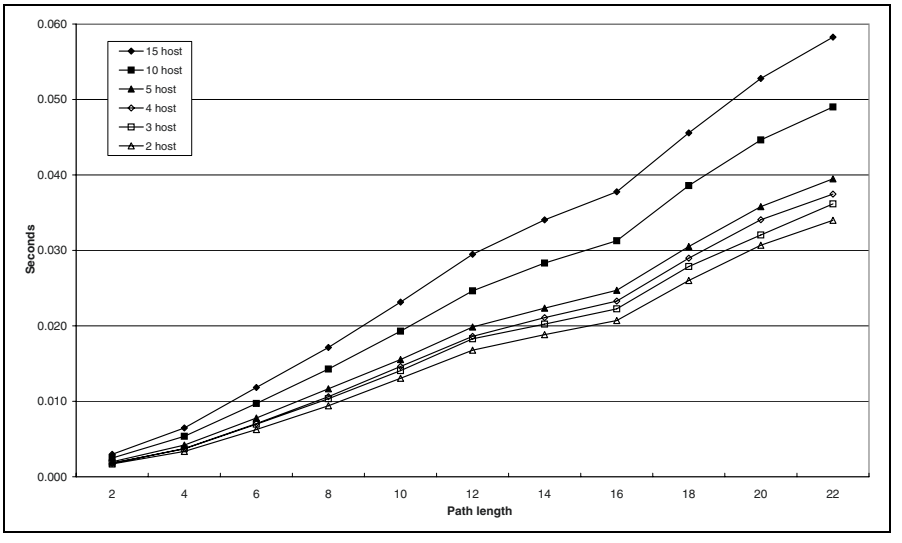


Fig. 4. Simulation of 100 task nodes partitioned across different numbers of hosts

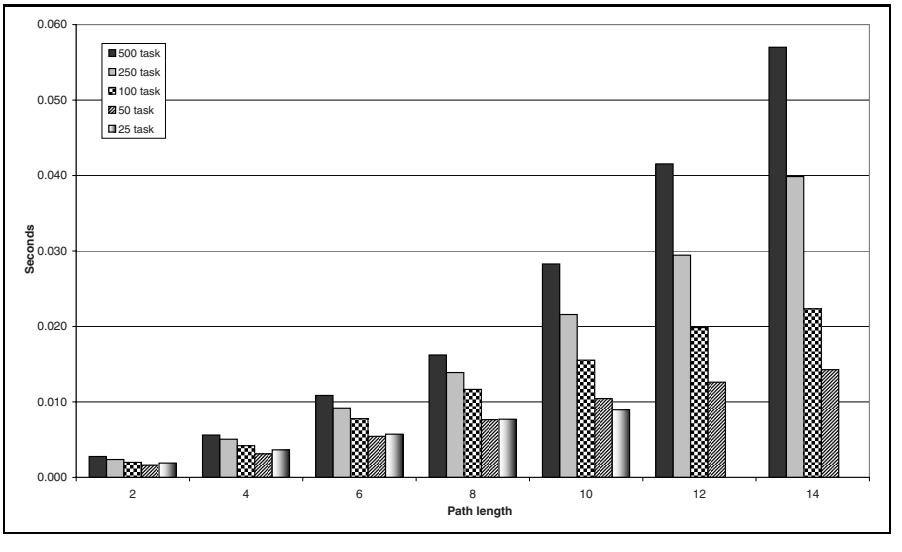


Fig. 5. Simulation of different numbers of task nodes partitioned across 2 hosts

running OS X 10.5.6 with a 2.33 GHz Intel Core 2 Duo processor and 2 GB of 667 MHz DDR2 memory. The third and fourth hosts were MacBook Pros running OS X 10.5.6 with 2.4G Hz Intel Core 2 Duo processors and 4 GB of 1067 MHz DDR3 memory. All hosts were running the Java 1.5.0_16 HotSpot Client VM. Connectivity among the hosts was verified before the measurements were started. The timing results for workflow graphs with 25, 50, and 100 task nodes are shown Figure 6.

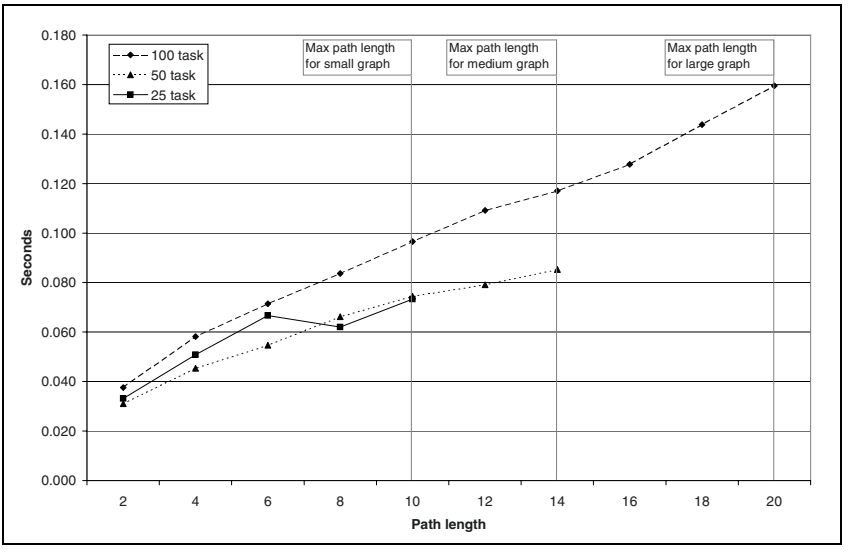


Fig. 6. Empirical performance of ad hoc wireless networking for different numbers of task nodes partitioned across 4 hosts

We can see from this graph that even in a realistic networking environment, our system shows the potential to solve large problems quickly. For example, even with a community knowledge base of one hundred tasks to explore, and a solution path length of twenty, our system finds and allocates a solution in under two tenths of a second on average.

5.1 Directions for Future Work

These encouraging results demonstrate that our system is ready to be evaluated against large-scale real-world problems. In order to accomplish this, we will seek a community to serve as a source of realistic benchmarks. We expect to face new issues when adapting our system to the rigors and challenges posed by our sample community.

One such concern for future research is the representation of tasks and specifications. Weakening our initial assumption that a specification only involves the inset and outset would allow specifications that include constraints on all

aspects of the workflow graph, such as path length, task preferences, and external temporal and spatial constraints. Furthermore, the specification can be expanded to influence the allocation and execution phases. A specification, for example, could minimize the set of participants or restrict the locations of certain tasks. In order to realize richer specifications, a more expressive formalism for describing tasks and preconditions and postconditions is necessary. For example, an extended formalism may allow associating variables and constraints with preconditions and postconditions such as type, capacity, and duration, or propagating constraints from one task to the next. As the sophistication of the formalism increases, more advanced planning techniques will come into play.

The handling of errors, community dynamics, and changes in the environment by the open workflow paradigm is another area for future research. For example, the allocation phase could wait indefinitely for a member with the needed capability and availability to join the community. During this time, an alternative workflow that avoids this resource limitation could be constructed. A failure during execution should result in a revised or repaired workflow, which requires reconstruction, reallocation, and compensating execution. Extending the current implementation with feedback mechanisms between the construction, allocation, and execution phases seems like a promising approach. Developing an appropriate commitment model that allows the participants to accomplish these activities in a mobile ad hoc setting is a focus for future work.

We also want to investigate relaxing the current restriction that construction and allocation are performed by a single host. A middleware that supports distribution of these tasks would allow construction and allocation in the face of fragmentation of the community and support localized recovery after a failure. When location constraints prohibit a rendezvous for data transfer, the system should be extended to consider scheduling participants into the workflow as couriers.

Finally, as with any application facing the rigors of the real world, security is critical. In addition to the usual concerns of trust, authorization, and privacy, the open workflow paradigm presents new challenges as it encourages participation across multiple administrative domains and social networks. Recognizing and handling changes in authorization and privacy due to roles and social context and resolving conflicting and competing specification ontologies are topics for future research.

6 Related Work

In this paper, we have focused on overcoming the challenges of bringing workflows to transient communities connected by mobile ad hoc networks. Standard workflow management systems, such as ActiveBPEL [6], Oracle Workflow Engine [7], JBoss [8], and BizTalk [9], 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., [10] removes the requirement of centralized control. Chafle, et al., [11], 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 [12] uses proxies for distributed control and has some design features that support mobile environments while Exotica/FDMC [13] describes a scheme to handle disconnected mobile hosts. In AWA/PDA [14], the authors adopt a mobile agent based approach based on the GRASSHOPPER agent system. WORKPAD [15] 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 [1] brings a full BPEL execution engine to a single cell phone, however that phone still acts as the sole coordinator. Finally, CiAN [2] 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 systems assume that a thoughtfully designed and fully specified workflow already exists. Open workflow is 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 [16], Workflow Prolog [17], and PDDL [18]. A review of further automated service composition methods may be found in [19]. Ponnekanti and Fox create workflows by rule-based chaining in SWORD [20], and discuss situations in which the resulting workflows may not produce the desired results due to the preconditions and postconditions of each task not being sufficiently specified. Fantechi and Najm [21] 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 simplified alternative to 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 have introduced the open workflow paradigm and presented the first algorithm for constructing open workflows in ad hoc wireless mobile

environments. A system for open workflow creation, allocation, and execution was proposed, implemented, and evaluated.

The open workflow paradigm is novel and enables 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 significant new challenges for the middleware, MANET, workflow, planning, and human-computer interaction research communities. The work presented in this paper is only the first step toward characterizing and addressing these concerns.

In producing the first practical implementation of an open workflow management system, we have affected a major paradigm shift in workflow middleware. Open workflows are much more than sophisticated scripts that enable one to exploit available services — they are a coordination vehicle for social and business activities that allows cooperating participants to construct and execute responses to needs identified by the participants. The open workflow paradigm enables the development of an entirely new class of systems that are nimble, mobile, and supportive of this new style of coordination.

Acknowledgments. This paper is based upon work supported in part by the National Science Foundation (NSF) under grant No. IIS-0534699. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF.

References

1. Hackmann, G., Haitjema, M., Gill, C., Roman, G.C.: Sliver: A BPEL workflow process execution engine for mobile devices. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 503–508. Springer, Heidelberg (2006)
2. Sen, R., Roman, G.C., Gill, C.D.: CiAN: A workflow engine for MANETs. In: Lea, D., Zavattaro, G. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 280–295. Springer, Heidelberg (2008)
3. Handorean, R., Gill, C.D., Roman, G.C.: Accommodating transient connectivity in ad hoc and mobile settings. In: Ferscha, A., Mattern, F. (eds.) PERVASIVE 2004. LNCS, vol. 3001, pp. 305–322. Springer, Heidelberg (2004)
4. Perkins, C.E., Belding-Royer, E.M.: Ad-hoc on-demand distance vector routing. In: WMCSA, pp. 90–100. IEEE Computer Society, Los Alamitos (1999)
5. Mobilab Group: Open workflow project web site, http://mobilab.wustl.edu/projects/openworkflow/
6. Active-Endpoints: ActiveBPEL engine, http://www.active-endpoints.com/active-bpel-engine-overview.htm
7. Oracle Inc.: Oracle workflow, http://www.oracle.com/technology/products/integration/workflow/workflow_fov.html
8. JBoss Labs: JBoss application server, http://www.jboss.com/docs/index
9. Microsoft Corp.: The BizTalk server, http://www.microsoft.com/biztalk/

10. Omicini, A., Ricci, A., Zaghini, N.: Distributed workflow upon linkable coordination artifacts. In: Ciancarini, P., Wiklicky, H. (eds.) COORDINATION 2006. LNCS, vol. 4038, pp. 228–246. Springer, Heidelberg (2006)
11. Chafle, G., Chandra, S., Mann, V., Nanda, M.G.: Decentralized orchestration of composite web services. In: Proc. of the 13th Intl. WWW Conference, pp. 134–143 (2004)
12. Sacramento, V., Endler, M., Rubinsztejn, H.K., Lima, L.D.S., Gonçalves, K., Bueno, G.A.: An architecture supporting the development of collaborative applications for mobile users. In: Proc. of WETICE 2004, pp. 109–114 (2004)
13. 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)
14. Stormer, H., Knorr, K.: PDA- and agent-based execution of workflow tasks. In: Proceedings of Informatik 2001, pp. 968–973 (2001)
15. 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. In: International Symposium on Technologies and Systems, pp. 173–180 (2006)
16. 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 (KR 2002), pp. 482–493 (2002)
17. Gregory, S., Paschali, M.: A prolog-based language for workflow programming. In: Murphy, A.L., Vitek, J. (eds.) COORDINATION 2007. LNCS, vol. 4467, pp. 56–75. Springer, Heidelberg (2007)
18. McDermott, D.: Estimated-regression planning for interactions with web services. In: Proceedings of the 6th International Conference on AI Planning and Scheduling, pp. 204–211. AAAI Press, Menlo Park (2002)
19. Rao, J., Su, X.: A survey of automated web service composition methods. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005)
20. 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)
21. Fantechi, A., Najm, E.: Session types for orchestration charts. In: Lea, D., Zavattaro, G. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 117–134. Springer, Heidelberg (2008)