TOWARDS A COMMON INFRASTRUCTURE FOR LARGE-SCALE DISTRIBUTED APPLICATIONS

C. Nikolaou [*], M. Marazakis [*], D. Papadakis [*], Y. Yeorgiannakis [*], J. Sairamesh [†]

Abstract

This paper discusses the requirements of current and emerging large-scale distributed applications and emphasises the need for a common infrastructure to support them. A design for an infrastructure that aims at satisfying these requirements is presented. Moreover, it is shown how key aspects of important large-scale applications can exploit the services included in the proposed infrastructure. The paper concludes by discussing the current status of a prototype implementation and our research plan.

1 Introduction

In this paper we consider a view of digital library systems that goes beyond that of a repository of on-line information, to provide support throughout all phases of work. As concluded in [Pae96], users need to find, analyze, and understand information of widely diverse types. They need to re-organize the information to use it in multiple contexts, and to manipulate it in collaboration with others. For this view to become reality, it is important to acknowledge that performing information-intensive tasks is a *process* that involves multiple phases at different levels of abstraction. This view requires substantial system support to provide *interoperability* among components, with an emphasis on collaboration and coordination, in the context of work

[1] Department of Computer Science, University of Crete and Institute of Computer Science, FORTH, PO Box 1385, GR 71110 Heraklion, Greece.
E-mail: {nikolau,maraz,dimpapa,georgian}@ics.forth.gr
[2] IBM Research, T. J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA.
E-mail: jramesh@watson.ibm.com

sessions. This is a major departure from assuming that the main objective of a digital library system is to locate a particular piece of information.

Such a shift of focus requires the introduction of collaboration and coordination support services as value-added services in current application platforms. Furthermore, the need for interoperability, which is the major requirement for successful integration of services in a large-scale distributed environment, leads to a need for a common infrastructure. Current state-of-the-art application development platforms, in particular distributed object frameworks such as CORBA [COR93, COR95] and DCOM [Cor96], do not provide such services. It is for this reason that more specialized frameworks with a strong emphasis on collaboration support are being developed. Examples include Netscape's ONE [ONE96], and DEC's LinkWorks WebWorker [Lin97].

From this perspective, current and emerging large-scale distributed applications that up to now have been considered in isolation can be expected to become integrated. Such applications include digital library systems, electronic commerce environments, scientific experiment management systems, distributed systems management environments, office automation workflow, and collaboration environments. A key requirement is the need for a communication, coordination, and collaboration infrastructure that can exploit the Internet as well as intranets to support user tasks. In such tasks, a collection of individuals and resources are combined, for a limited, but widely varying among applications, period of time, to accomplish a common goal. It is important to note that individuals and resources may have to be pulled from multiple organizational domains. Coordination is essential for accomplishing such tasks. The interactions involved in such applications can be viewed as processes involving multiple parties (individuals, organizations, resources), in a problem-solving distributed process.

A distinguishing feature of the applications under consideration is that they involve linking complete systems to accomplish user-defined tasks which require integration of individual capabilities offered by different systems. Capability integration can be achieved by composition of the available services. Unlike the layered approach, where a system is structured as a series of well-defined layers, with each layer using the services provided by the layer below and providing services to the layer above, a peer-to-peer interaction model appears to be a more natural fit for the applications under consideration.

A fundamental point of our approach is that these functions can be accomplished by a distributed workflow management facility. Workflows implement processes consisting of multiple, related processing steps which can be performed on or by heterogeneous processing entities

[KS95]. A process can be seen as a unit of work that flows through a network of "relevant" co-operating components so that services can be combined appropriately to achieve user objectives. In turn, this leads to a component-based paradigm for application development.

Viewing distributed applications as networks of cooperating components is consistent with developments in open, network-based technologies, such as the World Wide Web, distributed object frameworks and object request broker technology, office automation and collaboration support systems, message-oriented middleware to enable interoperation of existing applications, and mobile code technologies. Together with transaction processing and databases, these technologies provide a solid base for developing large-scale distributed applications.

In this paper, we propose a design for a common infrastructure based on the principles outlined, and discuss how key aspects of important large-scale distributed applications can exploit the proposed services. The architecture and components of the infrastructure are discussed in Section 2, which presents a design based on middleware services. Among these services, workflow management and mobile code technologies are of particular importance. In Section 3 it is shown how the proposed design can capture the requirements of distributed applications, by discussing a representative case study. Section 4 reviews related work in the area of providing services for interoperability and support for collaborative distributed applications. Finally, Section 5 concludes the paper, by discussing the current status of a prototype implementation and our research plan.

2 Workflow Model and Middleware Services Architecture

In the following we present a design for an infrastructure based on workflow management and agent technologies. Composition of services is possible only by providing a mechanism to identify and locate components that provide desired services, therefore the infrastructure is built upon a metadata repository service. The infrastructure must also provide support for process tracking. This includes keeping track of the progress of an interaction from start to completion, so that all interested parties can inquire about it, logging for auditing purposes, and performance monitoring. A workflow management service enables streamlining of (distributed) processes that consist of multiple activity steps. An agent execution environment is required to support the use of software agents that encapsulate user tasks.

These are *middleware services* [Ber96] which factor out commonly used functions into system components that can be shared across diverse hardware and software platforms. They are *generic* services, in that they provide support for common requirements for a wide range of applications. The architecture can also include application domain-specific services, which can be "value-added" services built on top of the basic services. Examples include support for payment, specialized directory services, customized process tracking services, and domain-specific "resource brokers". The overall architecture of the infrastructure is illustrated in Figure 1.

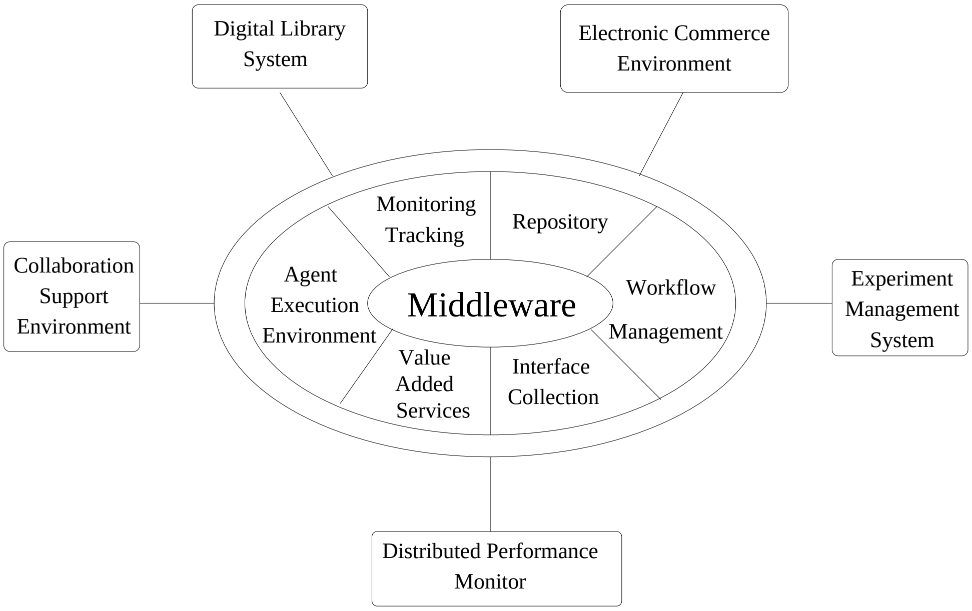

Figure 1: Middleware Services

2.1 Specification Language and Repository Service

Managing the information that describes the different components of a large-scale distributed system is an essential part of the proposed middleware services architecture. This information is managed by a network of repository systems [BD94] which are integrated in a global repository service, which allows components to share data without knowledge of other tools' internals. The repository service associates names with information objects. This information includes descriptions of resources, tasks, and events, expressed using the constructs of a specification language that we are developing.

Resources include systems hardware, applications (such as database systems), as well as human workers (represented by the interaction tools that they use). Resource descriptions are in the form of attribute-value pairs. A resource's attributes include mainly information required for accessing the resource and authorization information. It is also possible to describe the resource's configuration, in other words its component parts. They may also include information about supported quality of services levels.

A task can be either *simple*, in which case it represents a single activity step that requires a single resource, or *compound*, in which case it involves multiple steps with control and data flow interdependencies. It is possible to associate attribute-value pairs for tasks, in the same manner as for resources. This is essential to enable dynamic binding of tasks to resources, via a *matchmaking* process. This can also serve as the basis for building complex negotiation protocols. A simple task's attributes include information about the input and output of the task, and information about the resource to be used for performing the task. The attributes of a compound task include a specification of the flow of control and data required to implement the task's objective. This specification is in terms of a flow of results from each simple task in the workflow to others that depend on it. Each task is associated with an input and an output communication port, through which data flow occurs. At run-time, an output port can be connected to one or more input ports.

The specification language allows the definition of *groups* of simple tasks. A task group represents a set of tasks with *dynamic* membership. All members execute the *same* task. The actual members of a task group are *not* known at process definition-time. Each task group has an input and an output port, which, at run-time, are connected to the corresponding ports of each task group member. The group construct facilitates modeling workflows where a task interacts with an arbitrary number of other tasks that all perform a similar function. Thus, the group construct allows modeling of interactions in terms of *roles*, rather than in terms of one-to-one interactions. From a process modeling point of view, a group is treated as a single entity, with its own state transitions and communication ports, while hiding all implementation-level information that only becomes available at run-time. An example of using the group construct is given in Section 3.

Flow of control and data is conditional on the occurrence of certain events, which are specified in the definition for the compound task. Event occurrence is combined with the evaluation of *transition conditions* in event-condition-action rules [DHL93] to trigger flow of control events (such as task start). Flow of data is through communication channels which are setup to connect the output port of a task to the input port of one or more tasks. The establishment of such channels is coupled with flow of control events.

Events of particular interest are the changes of state of executing tasks. Each task is required to notify its associated task management component when a significant state transition is to take place, particularly when the task begins execution (`START` transition), and when it terminates either successfully ((`DONE` transition) or due to an exception (`FAIL` transition). Optionally, a task can provide notification of the beginning and end of *phases* during its execution. The definition of phases is application-dependent. Phases can be nested, thus providing the ability to monitor task status in several levels of detail. This is particularly important for monitoring purposes. An event may have associated with it a set of attribute-value pairs, which are passed - uninterpreted - by the event dispatch mechanism to "event consumers" when an instance of the event occurs. Event management is handled by the monitoring infrastructure, discussed in Section 2.4.

The information managed by the repository service is expressed in a specification language, using constructs for the description of resources, events, simple tasks, and compound tasks (workflows). The attributes specified for an object are completely user-defined. Attribute-value pairs can be defined for resources, tasks, and events to support specific applications. This is a mechanism for expressing application domain-specific information that is required for interoperability. Values can be either of primitive data types (such as string and integer) or of aggregate data types (such as arrays and lists). It is also possible to use names of objects as values. This allows the description of resources as aggregations of component resources, and the definition of workflows as compounds of simple tasks and other workflows.

The repository service exports an API which can be used by other components of the infrastructure as well as by user-defined tasks. A parser for the specification language uses this API to provide resource and task descriptions to the repository service. By providing persistent storage of definitions, it is possible to perform *incremental updates* of the repository, via API operations that add, change, and delete objects as well as attribute/value pairs.

The repository service is structured as a set of cooperating but autonomous servers that collectively maintain the metadata that is required by the infrastructure and the applications built on top of it. When a server cannot answer a query, it can either forward it to another server, in the same way as DNS, or respond to the requesting client by indicating the server to which the query is to be forwarded. In order to sustain high load and system failures, and to reduce network bandwidth consumption, it is necessary to support replication of repository servers. This can be achieved by organizing replicas as groups organized in a master/slave configuration. For efficiency, updates may be collected in batches and propagated periodically.

2.2 Workflow Management

Workflow automation technology [GHS94, KS95, SGJ+96] offers a framework for the construction of complex application systems without hard-coding control flow dependencies in applications. This is particularly important for applications which are structured as a collection of cooperating components with dynamic interconnections.

Components can be developed in a modular fashion, so as to provide specific services to units of work that flow through the system, representing actual business processes, and then be dynamically linked together as needed. The desired flow of data and control is described (and therefore documented) as a process model to the workflow management service, using the constructs provided by the specification language. At execution time, for each process instance that is initiated, the workflow scheduler sequences the activity steps, passing data and control appropriately among components. Removing assumptions (which often are implicit and undocumented) about program sequencing and data routing from the code of application programs greatly improves their robustness to changes.

The workflow scheduler translates workflow definitions into a sequence of commands which coordinate the establishment of a workflow instance-specific network of cooperating components, and then proceeds to activate the tasks of the workflow that have no predecessors. The information required for sequencing the execution of a workflow instance is distributed among task management components, rather than having a single scheduling component coordinate execution. This is a departure from the design of current commercial workflow management systems, which centralize scheduling and control. Other research projects investigating decentralized workflow scheduling include Exotica/FMQM [AAA+95] and WIDE [CGS97].

Each task management component is responsible for executing an activity step of the workflow, which is modeled as a single *simple task*, involving access to a single resource. Each such component is informed by the workflow scheduler about its *activation dependencies*, which determine when the actual task can start execution. These dependencies are expressed (using the specification language) as boolean conditions involving events related to changes in task states, as well as system-related events and application-defined events. When all activation dependencies are satisfied, the task management component activates its associated task. A software agent can encapsulate both the task management component and its associated task in a single entity. While the task remains active, it can retrieve data from its input port and forward data to its output port. Input and output are in the form of *data streams*, which represent a continuous flow of data from a producer task to one or more consumer tasks. Task execution may generate events of interest to other tasks. A task's duration may vary depending on the nature of the process in which it participates.

2.3 Agent Technology

For our purposes, *agents* are encapsulations of executable code and run-time state that can execute asynchronously. Agents can be used to compensate for differences among systems in terms of interaction models and data formats. Agents encapsulate simple tasks, accessing a single resource, in the context of compound tasks with multiple activity steps, that require access to multiple, heterogeneous resources.

An agent can access a resource by dynamically loading the resource's interface code. This allows extending the functionality of an agent on-demand. A monitoring infrastructure keeps track of resource accesses (see Figure 2).

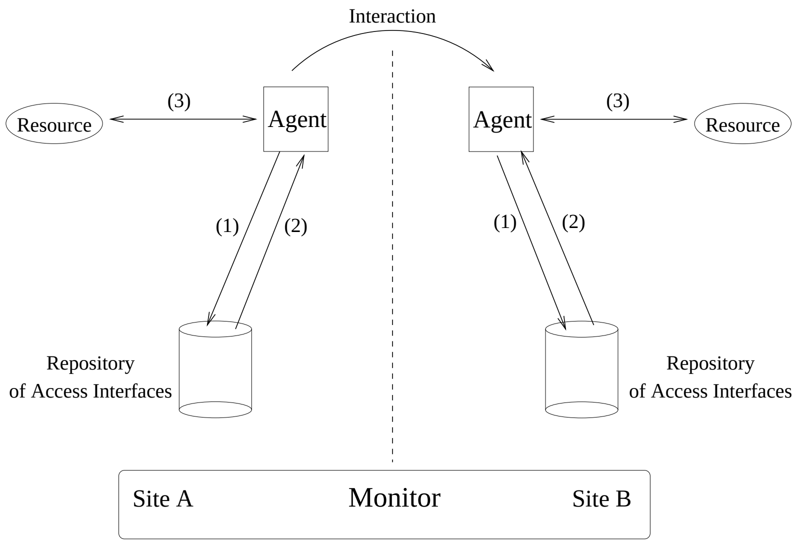

Figure 2: Agents in a Distributed Workflow.

2.4 Monitoring Service

Distributed environments such as the ones discussed in this paper are composed of any number of distinct, heterogeneous components. What is more, there might be frequent changes in the components and their interconnections. In order to support management actions, applications and resource managers need to provide information about their structure and internal state. Applications should be instrumented to monitor events of relevance to the resources they use. Such instrumentation is necessary to collect management-related data and events, and forward them to specialized management applications, possibly after filtering.

Task management components provide a management interface which allows the monitoring service to track the progress of units of work, retrieve task-specific instrumentation data, and record changes of task state. Maintaining context and status information enables run-time monitoring and control of processes. Context and state information can be made persistent, thus enabling exception handling, that can entail process roll-back or process restart, as well as the execution of compensating tasks.

Resource description using the specification language discussed in 2.1 allows us to specify what status information is made available by an application component and how a management tool can obtain this information. Status information includes the metrics (measurements) that can be requested from an application component, and the events (with their associated attributes) that a component can generate. Thus, the repository service can provide both descriptive information about a component and information about management interfaces or tools that enable monitoring and control of the component.

Keeping track of agents' activities is essential for monitoring and control of workflows. This is achieved by requiring each agent to encapsulate the management logic for a task, and to register with a distributed monitoring infrastructure. Using the management information maintained by the repository service regarding the resource (application component) that is to be accessed, the monitoring facility can retrieve status information from agents. This in turn enables management applications to correlate the individual steps of a unit of work.

3 Case Study

Performing work in a generalized digital library environment is a distributed process that may entail activities such as searching and advertising, negotiating, contracting and ordering, billing and payment, distribution and receipt, and other customer services. Not all these activities are performed in every transaction, nor are they necessarily performed in this order. Indeed, they may be performed in parallel [Sch96]. In this section we illustrate how the infrastructure presented in Section 2 satisfies the requirements of such a distributed work environment, by examining how an interaction scenario can be realized using the proposed services.

Consumers of digital information that explore the Web for new resources can use agents that learn their preferences, maintain profiles, seek out appropriate providers, and when necessary request further information. *Collector* agents contact *broker* agents requesting information about products that satisfy consumer preferences. This information can be further processed by *evaluator* agents that use more sophisticated techniques to filter and evaluate collected information. Then *decision* agents may be used to alert consumers about products of potential interest or proceed to purchases. Purchases require *negotiation* agents to agree with providers on prices, billing, payment, and delivery options. Finally, *delivery* agents can deliver purchased products to consumers, collect payment, and enforce intellectual property rights.

Performing these activities requires collaboration of a variety of agents and co-ordination of information flow among them. As a representative example, we consider a search scenario involving collector, evaluator, broker, and provider agents. In Appendix A we show sample resource, task, and task group definitions, using our specification language, for this interaction scenario. Appendix B describes the behavior of the broker, collector, and evaluator agents using pseudo-code.

The *collector* task is initialized with some consumer's profile that contains its preferences as well as a collection of supported methods to access providers. This task consists of two phases. An information broker agent is contacted that matches the consumer's preferences with a list of providers that may satisfy these preferences; then providers are accessed and their collections are examined. Communication between *collector* and *broker* is established by the workflow manager as described in the specification of the search workflow. Therefore, neither the *collector* nor the *broker* should explicitly manage connections. They simply interact with their standard input and output communication streams.

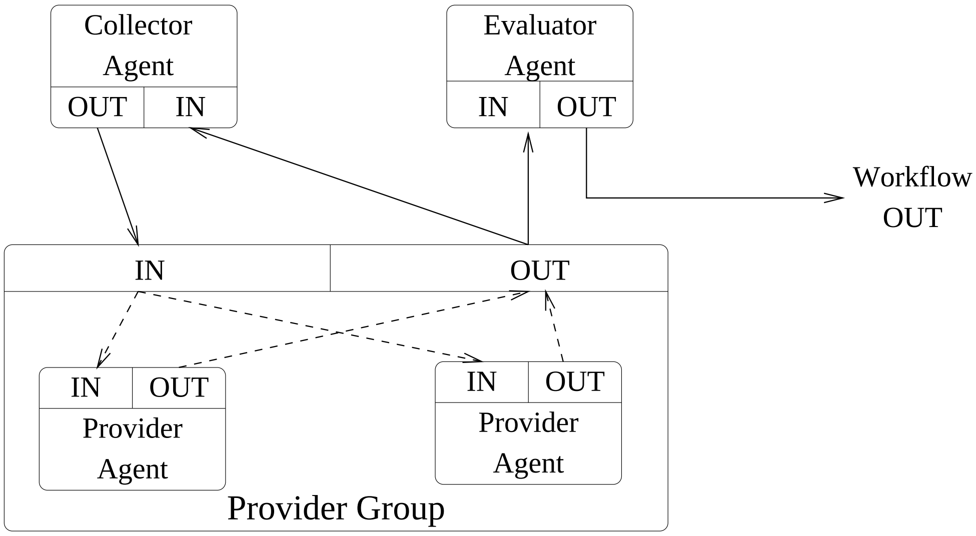

Figure 3: Search Scenario: Interaction between Collector, Evaluator, and Provider Group.

Completion of the *broker interaction* phase is an event that triggers the workflow manager to enable the *access providers* phase, where collaboration and co-ordination among *collector, evaluator*, and a *provider group* is required. Figure 4 shows the description of the search workflow using the specification language that we are developing. The specification clearly marks the start and end of the "broker interaction" and "provider interaction" phases. `ProviderTaskGroup` is the task group corresponding to the set of providers to be contacted. Figure 3 illustrates the information flow between the collector, the evaluator, and a group of providers, as described in the specification language and established by the workflow manager for the "provider interaction" phase.

```
DEFINE WORKFLOW SearchScenario –
  TYPE: StructuredProcess;
  INPUT: –SearchKeywords: STREAM – Keyword: STRING; Weight: INTEGER;˝;
        Broker: OBJECT; Repository: OBJECT;˝;
  OUTPUT: STREAM – FilteredResult: OBJECT; ˝;
  PLAN: –
    # Upon workflow activation ...
    WHEN (SearchScenario.START) –
      Collector.START [ SearchScenario.Input BECOMES Collector.Input ];
    ˝
    WHEN (Collector.START) –
      # BrokerTask must get Collector's search criteria ...
      Collector.START˙PHASE("Broker Interaction");
      BrokerTask.START [ Collector.Output BECOMES BrokerTask.Input;
                   BrokerTask.Output BECOMES Collector.Input ];
      Collector.END˙PHASE("Broker Interaction");
    ˝
    WHEN (Collector.END˙PHASE("Broker Interaction")) –
      Evaluator.START [ Collector.Input BECOMES Evaluator.Input;
                  Evaluator.Output BECOMES SearchScenario.Output];
      Collector.START˙PHASE("Access Providers") [
        ProviderTaskGroup.Output BECOMES Collector.Input;
        Collector.Output BECOMES ProviderTaskGroup.Input ];
    ˝
    WHEN (Collector.END˙PHASE("Access Providers")) –
      Collector.DONE;
      ProviderTaskGroup.DONE;
      Evaluator.DONE;
      SearchScenario.DONE;
    ˝
      ˝
  ATTRIBUTES:
    Description: "Workflow corresponding to the search scenario";
    Participants: –"Collector", "Evaluator", "Broker", "ProviderTaskGroup"˝;
˝
```

Figure 4: Search Workflow (expressed in the proposed specification language).

A variety of diverse methods and formats is used to maintain information sources, therefore a collection of access interfaces is maintained in the repository, as well as descriptions about the access methods supported by providers. A matchmaking procedure is used to select an access method that both the *collector* and the provider understand (see Figure 2). If such a method does not exist, then the *collector* fails notifying the workflow manager to terminate the interaction. When the providers receive from the *collector* the consumer's preferences they reply with information about products of potential interest that flows to the *evaluator* (figure 3). As in the broker interaction case, the communication channnels required are established by the workflow management service.

The number of providers participating in this interaction is not known beforehand. The specification language allows us to model this aspect, by defining the interaction in terms of communication channels between the collector, the evaluator, and a *provider task group* (see Figure 3).

4 Related Work

This section reviews related work in the area of providing services for interoperability and support for collaborative distributed applications. Related work in the area of languages for workflow specification is also reviewed.

The Stanford Digital Library Project [PCGM+96] focuses on the issue of integration, by structuring a digital library system as a collection of components that export well-defined interfaces and rely on a set of common services to perform distributed system functions like communication. The InfoBus testbed applies distributed object technology to enable interoperability according to these principles, by using wrapper objects to present a unified interface of digital library services. In our design, interoperability is achieved by having agents *dynamically* load the access interface of resources.

In the University of Michigan Digital Library [ABD+96], interacting software agents cooperate and compete for resources in the context of a virtual information economy to provide library services to users. Each agent performs a highly specialized library task and has a generic communication interface, and represents either a resource or a functional unit of the overall system. In order to service user requests, functional units and resources need to be dynamically combined, by forming *agent teams* under the guidance of query planning agents. An essential aspect of this approach is that the capabilities and requirements of any functional unit in the digital library system are explicitly described using a formal language. Using this facility, it is possible for agents to negotiate about provision of complex services as well as resource allocation. In our approach, the specification language allows explicit description of capabilities and requirements, and, moreover, of execution plans for compound tasks.

[Sch96] argues for a common infrastructure to support both digital libraries and electronic commerce, as they have common requirements to support activities such as searching, advertizing, negotiation, contracting, billing, payment, production, distribution, accounting, customer services, and information management. It is our view that a common infrastructure could support other large-scale distributed applications as well.

[Pae96] illustrates, through a discussion of two case studies regarding user requirements for a digital library system, that searching is not enough for a digital library system to go beyond the functionality offered by existing information retrieval systems. This is further discussed in [Lar97], which challenges some of the underlying assumptions that have historically guided the development of networked information systems, and particularly raises the issue of a paradigm shift regarding searching for information. The conclusion from [Pae96] is that it is essential to support users in multiple phases of their work, including information discovery, retrieval of information artifacts, interpretation of retrieved "raw" information content, management of information, and sharing results with coworkers. Our view is that such activities are a natural fit for a distributed workflow infrastructure.

A similar view is taken in [CCD+96, Cha96]. In [CCD+96] a Java-based system that supports peer-to-peer communication among distributed processes is described. The focus of this work is on identifying and specifying software components that can be composed to create distributed applications, layered on top of standard network technologies such as the WWW. This is further elaborated in [Cha96], which gives an overview of the Caltech Infospheres Project. This project is concerned with managing *virtual organizations*, which are "fluid organizations that link different groups of people and resources temporarily to carry out missions as they arise".

A distinguishing feature of our approach is the use of a specification language to describe both resource capabilities and task requirements, and, moreover, define the control and data flow of compound tasks. Related work places the emphasis on multidatabase systems, with a focus on models of task dependencies. The specification language presented in this paper not only provides for task dependencies specification but also offers a component interconnection paradigm for workflows. Moreover, by supporting the notion of *task groups*, it is possible to model more dynamic interaction scenarios than languages that define dependencies in terms of explicit task instances. Another distinguishing feature is the ability to define application-specific events, corresponding to the beginning and end of *phases*.

[BCE+93] describes the InterBase prototype, which supports a flexible transaction model that allows *global transactions* to execute over heterogeneous database systems. Global transactions are specified using the InterBase Parallel Language (IPL), as subtransactions with control and data flow dependencies. InterBase automatically executes subtransactions in parallel if this is possible without violating dependencies.

[KS95] presents a workflow specification language (WFSL) to model process flow and a task specification language (TSL) to provide an interface to heterogeneous, autonomous, and distributed systems (mainly databases), thus separating workflow design from the details of interacting with applications. WFSL allows a declarative rule-based description of control and data flow. An important point in [KS95] is a detailed task dependency model which allows management of atomicity and consistency of workflows, by capturing the behavior of both transactional and non-transactional tasks.

5 Concluding Remarks

We are building the infrastructure discussed in this paper and we plan to use it as the basis for developing prototypes of three major types of applications: environment for collaboration, electronic commerce, and distributed systems management. A prototype is being developed as an extensible framework for building collaborative systems.

We have implemented the parser for the workflow specification language and a centralized repository service, where objects specified by the workflow language are stored. This repository implementation will be enhanced to support replication and distribution. A primitive agent environment is also implemented in Java. Mobile agents can migrate to sites to access remote resources through a collection of access interfaces that is maintained in the repository. An appropriate network loader has been also implemented to enable agents to load access interfaces from the network, if they are not available locally. The loader also provides to agents the means to examine interfaces and extract methods to invoke as well as their arguments. Finally, an interface to Dienst have been implemented so that agents can send requests to and receive responses from Dienst servers. Our collection will be enriched with access interfaces to various search engines and other resources (such as databases).

Our research plan includes an investigation of agent negotiation. If providers are pricing their information, then agents receive more information about prices, billing, payment, and delivery options. Providers can utilize agents to negotiate purchases with consumer agents. Furthermore, provider agents may actively seek consumers searching their profiles, informing them about new releases, possible discounts and new options. The same interactions models can be applied to manage large-scale distributed systems. Agents that represent resources interact with other that seek out idle resources, negotiate billing, pricing, and submission of jobs. We are interested in developing models of such interactions, using our specification language.

References

[AAA+95] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, M. Kamath, and R. Guenthoer. "Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management". In *Proc. IFIP Working Conference on Information Systems Development for Decentralized Organizations*, 1995.

[ABD+96] D.E. Atkins, W.P. Birmingham, E.H. Durfee, E.J. Glover, T. Mullen, E.A. Rundensteiner, E. Soloway, J.M. Vidal, R. Wallace, and M.P. Wellman. "Toward Inquiry-Based Education Through Interacting Software Agents". *IEEE Computer*, pages 70–76, May 1996.

[BCE+93] O. A. Bukhres, J. Chen, A. K. Elmagarmid, X. Liu, and J. G. Mullen. "InterBase: A Multidatabase Prototype System". In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1993.

[BD94] P. A. Bernstein and U. Dayal. "An Overview of Repository Technology". In *Proc. 20th VLDB Conference*, 1994.

[Ber96] P.A. Bernstein. "Middleware: A Model for Distributed System Services". *Communications of the ACM*, 39(2):86–98, February 1996.

[CCD+96] K.M. Chandy, A. Chelian, B. Dimitrov, H. Le, J. Mandelson, M. Richardson, A. Rifkin, P.A.G. Sivilotti, W. Tanaka, and L. Weisman. "A World-Wide Distributed System Using Java and the Internet". In *Proc. 5th IEEE Int'l Symposium on High Performance Distributed Computing*, 1996.

[CGS97] S. Ceri, P. Grefen, and G. Sanchez. "WIDE: A Distributed Architecture for Workflow Management". In *Proc. 7th Int'l Workshop on Research Issues in Data Engineering*, 1997.

[Cha96] K.M. Chandy. "Caltech Infospheres Project Overview: Information Infrastructures for Task Forces", November 1996. Available via URL http://www.infospheres.caltech.edu.

[COR93] *"The Common Object Request Broker: Architecture and Specification"*. Object Management Group, Framingham, Mass., 1993.

[COR95] *"Common Facilities Architecture"*. Object Management Group, Framingham, Mass., 1995.

[Cor96] Microsoft Corp. "DCOM Technical Overview", 1996. Available via URL http://www.microsoft.com.

[DHL93] U. Dayal, M. Hsu, and R. Ladin. "Organizing Long-Running Activities with Triggers and Transactions". In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1993.

[GHS94] D. Georgakopoulos, M. Hornik, and A. Sheth. "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure". *Distributed and Parallel Databases*, 3(2), September 1994.

[KS95] N. Krishnakumar and A. Sheth. "Managing Heterogeneous Multi-System Tasks to Support Enterprise-Wide Operations". *Distributed and Parallel Databases*, 3(2), April 1995.

[Lar97] R. L. Larsen. "Relaxing Assumptions - Stretching the Vision: A Modest View of Some Technical Issues". *D-Lib Magazine*, April 1997.

[Lin97] "LinkWorks Software Product Description", 1997. Available via URL http://www.digital.com/info/linkworks.

[ONE96] "The Netscape ONE Development Environment Vision and Product Roadmap", 1996. Available via URL http://home.netscape.com/comprod/one.

[Pae96] A. Paepcke. "Digital Libraries: Searching is not Enough". *D-Lib Magazine*, May 1996.

[PCGM+96] A. Paepcke, S. B. Cousins, H. Garcia-Molina, S. F. Hassan, S. P. Ketchpel, M. Roscheisen, and T. Winograd. "Using Distributed Objects for Digital Library Interoperability". *IEEE Computer*, 29(5):61–68, May 1996.

[Sch96] D. Schutzer. "A Need for a Common Infrastructure: Digital Libraries and Electronic Commerce". *D-Lib Magazine*, April 1996.

[SGJ+96] A. Sheth, D. Georgakopoulos, S. Joosten, Rusinkiewicz, W. Scacchi, J. Wilden, and A. Wolf. Report from the NSF Workshop on Workflow and Process Automation in Information Systems. Technical report, University of Georgia, 1996. Available via URL http://LSDIS.cs.uga.edu/publications.

Appendix A

```
# Search workflow (see Section 3) expressed in our specification language
**RESOURCE DEFINITION SECTION**
DEFINE RESOURCE BrokerResource {
    TYPE : BrokerService;
    ATTRIBUTES:
        # Runtime requirements
        Arch : "ALPHA"; OpSys : "DEC OSF1"; OpSysVer : "4.0";
        ExecMode : native;
        # Initialization Information ...
        Path : "/usr/bin/broker";
        Arguments : "-f /etc/broker.config";
        # Broker can be accessed via :
        AccessProtocols : {"telnet", "http"};
        # protocol-specific parameters:
        #  telnet: port, login, password, http: port, URL
        Telnet : { 23, "brokerService", "" };
        Http : { 80, "www.csd.uch.gr/cgi-bin/brokerService"};
        # Access Control Information
        Acl : { ["all", ["start", "stop"]],
                ["free", ["all", "limitedSearch"]],
                ["subscriber", ["free", "fullSearch"]],
                ["sysadm", ["subsrciber", "inspect"]] };
}
**TASK DEFINITION SECTION**
DEFINE TASK Collector {
    TYPE  : InterpretedAgent;
    # Input is a variable set of keywords (string)
```

```
    #        weight (integer) pairs
    #        and the broker, repository service handles
    INPUT : {SearchKeywords: STREAM { Keyword: STRING; Weight: INTEGER;};
            Broker: OBJECT; Repository: OBJECT;};
    OUTPUT : STREAM { Location: STRING; Result: OBJECT; };
    ATTRIBUTES:
        # Runtime requirements
        Arch : ""; OpSys : ""; OpSysVer : "";
        ExecMode : interpreted;
        Interpreter : "Java"; Version : "JDK 1.1";
        # Access Control Information
        Acl : { ["all", ["collectorInit", "collectorStop"]],
                ["sysadm", ["all", "collectorInspect"]] };
        # Activation Information
        # Path = Location of the Interface Collection Repository
        Path : "http://www.ics.forth.gr/JavaClass/Repository";
        # main() method of Collector agent
        ActivationMethod : "collectorInit";
}
DEFINE TASK BrokerTask {
    TYPE  : InterpretedAgent;
    # Input: a variable set of keyword-weight pairs, plus handles for the
    #        broker and repository services
    INPUT : {SearchKeywords: STREAM { Keyword: STRING; Weight: INTEGER;};
             Repository: OBJECT;};
    OUTPUT : STREAM { Location: STRING; };
    ATTRIBUTES:
        # Runtime requirements
        Arch : ""; OpSys : ""; OpSysVer : "";
        ExecMode : interpreted;
        Interpreter : "Java"; Version : "JDK 1.1";
        # Access Control Information
        Acl : { ["all", ["brokerInit", "brokerStop"]],
                ["sysadm", ["all", "brokerInspect"]] };
        # Activation Information
        # Path = Location of the Interface Collection Repository
        Path : "http://www.ics.forth.gr/JavaClass/Repository";
```

```
        # main() method of Collector agent
        ActivationMethod : "brokerInit";
}
DEFINE TASK Evaluator {
        TYPE  : InterpretedAgent;
        # INPUT is the OUTPUT of Collector Agent
        # OUTPUT is the filtered results of collector
        INPUT : STREAM { Location: STRING; CollectorResult: OBJECT; };
        OUTPUT : STREAM { FilteredResult: OBJECT; };
        ATTRIBUTES:
        # Runtime requirements
        Arch : ""; OpSys : ""; ExecMode : interpreted;
        Interpreter : "Java"; Version : "JDK 1.1";
        # Access Control Information
        Acl : { ["all", ["evaluatorInit", "evaluatorStop"]],
                ["sysadm", ["all", evaluatorInspect]] };
        # Activation Information
        # Path = Location of the Interface Collection Repository
        Path : "http://www.ics.forth.gr/JavaClass/Repository";
        # main() method of Evaluator agent
        ActivationMethod : "evaluatorInit";
}
DEFINE TASK ProviderTask {  # executed by each provider in ProviderTaskGroup
 .............................................................................
}

**TASK GROUP DEFINITION**
DEFINE TASK_GROUP ProviderTaskGroup {
   TYPE: StructuredProcess;
   INPUT: {SearchKeywords: STREAM { Keyword: STRING; Weight: INTEGER;};
            Broker: OBJECT; Repository: OBJECT;};
   OUTPUT: STREAM { FilteredResult: OBJECT; };
   ATTRUBUTES:
      # At run-time, all members execute an instance of ProviderTask ...
      TASK: ProviderTask;
 .............................................................................
}
```

Appendix B

In this appendix, the *broker, collector, evaluator* agents of the search scenario discussed in Section 3 are described in pseudo-code. Instrumentation, in the form of procedure calls such as `START(), DONE(), START_PHASE(), END_PHASE()`, notifies the workflow scheduler of events related to the execution state of the workflow. All interactions among the agents are in terms of `read` and `write` operations on their input and output ports (`in` and `out`, respectively). Communication channels among these ports are established by the workflow manager, according to the specification in Figure 4.

```
Agent BrokerTask() {
    START();
        in.read(SearchingCriteria);
        ProviderList = Repository.query(SearchingCriteria);
        out.write(ProviderList);
    END();
}
Agent Collector(SearchingCriteria:STREAM, SupportedAccessMethodList:OBJECT) {
    START();
    START_PHASE("Broker Interaction");
        out.write(SearchingCriteria);
        in.read(ProviderList:OBJECT);
    END_PHASE("Broker Interaction");
    START_PHASE("Access Providers");
        for each Provider in ProviderList {
            AccessMethodList = Repository.query(Provider, "AccessProtocols");
            AccessMethod = match(AccessMethodList, SupportedAccessMethodList);
            if AccessMethod == NULL
               FAIL();
            NetworkLoader.loadAccessInterface(AccessMethod);
            AccessMethod.out.write(SearchCriteria);
            // in.read(Result) is redirected to Evaluator Agent
        }
    END_PHASE("Access Providers");
    DONE();
}
Agent Evaluator() {
    START();
```

```
        AcceptCriteriaList:OBJECT
        in.read(ProviderProductList);
        for each Product in ProviderProductList {
            action = EvaluationStrategy(Product, AcceptCriteriaList);
            if action == ACCEPT
                out.write(Product);
        }
    DONE();
}
```