

Workflow management systems on top of OSF DCE and OMG CORBA

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

1996 Distrib. Syst. Engng. 3 250

(<http://iopscience.iop.org/0967-1846/3/4/005>)

View [the table of contents for this issue](#), or go to the [journal homepage](#) for more

Download details:

IP Address: 38.107.179.213

The article was downloaded on 20/02/2012 at 21:02

Please note that [terms and conditions apply](#).

Workflow management systems on top of OSF DCE and OMG CORBA

Alexander Schill^{†§} and Christian Mittasch^{‡||}

[†] Fakultät Informatik, TU Dresden D-01062 Dresden, Germany

[‡] Institut für Informatik, TU Bergakademie Freiberg D-09596 Freiberg, Germany

Abstract. This paper describes a new workflow management approach and system named CodAlf. It is based on a decentralized system architecture. As compared to existing research-oriented and commercial solutions, the approach especially enables the support of distributed organizations as well as dynamic growth and reconfiguration. The implementation uses the OSF Distributed Computing Environment (DCE) as a basis. All components are modelled and implemented as objects of a distributed system on top of DC++, an object-oriented DCE extension. In addition to runtime support, a workflow specification language and a workflow design tool are offered. The system is also compared to a CORBA-based approach named BPAFrame, a very recent development within our group. Experiences with typical applications are reported.

1. Introduction

Workflows are found in any organization. A workflow consists of a number of tasks to be performed by several participants. While workflows can be executed manually, partial automation can enhance productivity and can enable the use of computerized tools for the various tasks. A workflow management system supports this goal: it coordinates the user and system participants, together with the appropriate data resources to achieve defined objectives by set deadlines. The coordination involves passing tasks from participant to participant in correct sequence, ensuring that all fulfil their required contributions [16]. At least, an organization has to be modelled as a triple of organizational structure, resources (hardware, software, employees) and processes (realized by workflows), see also [35].

Many earlier research approaches addressed this problem; examples are Action Paths [2], PAGES [10], EuroCoop [13], IPSO [33], Mentor [41], Meteor [19], WASA [40], Exotica [21], Mobile [4, 16] and Income/Star [26]. These systems resulted in basic prototypes but hardly provided language and tool support. More recently, several workflow management products have emerged; some examples are COSA [7], FlowMark [9], WorkParty [39], Exotica [21] and Prominand [32]. These systems support comfortable workflow specification techniques and enable the integration of existing standard applications such as text processing or spreadsheets. However, they typically use a centralized database approach for implementing workflow management at runtime. This leads to well known problems of single points of failure, to potential performance bottlenecks, and also to a lack of scalability

and dynamic reconfigurability. Most systems also depend on proprietary communication protocols.

In the meantime, standardization efforts have been initiated and are conducted by the Workflow Management Coalition (WfMC) [37, 38], a consortium of hardware vendors, software developers and end users. This organization has proposed a general workflow system architecture that consists of process definition tools, administration and monitoring tools, workflow enactment services (runtime services) and application software. Moreover, a workflow specification language is being standardized. However, the standardization is still under discussion and has not been completed yet.

Current projects address the lack of the mentioned research approaches and available products. The research focuses are as follows

- decentralized control: CodAlf [22] (as discussed here), BPAFrame [23] and Exotica/FMQM [1], Mobile [17] and Meteor2 [24],
- decentralized data: Exotica/FMQM, Mentor [41], Joosten [18] and Meteor2 [19, 24],
- object-oriented extensions: Mentor, ActionPaths, WIDE-project [6], Mobile and Meteor2,
- applications of current software engineering and consequently new design within an overall modelling of business objects: BPAFrame, Joosten, Meteor2 and Mobile [3, 5]. Here the first commercially available system can be found as ‘component-ware’,
- combinations with extended transaction processing, for instance Mentor, Meteor2,
- and extensions to Internet integration, e.g. via www in Meteor2.

Based on this background, this paper describes CodAlf (‘Code Name Alfa’), a new, decentralized approach and

[§] E-mail address: schill@ibdr.inf.tu-dresden.de

^{||} E-mail address: chris@ibdr.inf.tu-dresden.de

system towards distributed workflow management. As opposed to most existing solutions, this especially enables the support of distributed organizations as well as dynamic growth and scalability. One of the focus areas is on reliability realized by extended error detection and recovery.

Most existing systems are concentrated on central servers. Thus, they only support workflow management within that department or organization pursuing that server. Our approach supports autonomous work in each department. Furthermore, it supports both growth and interactions with other autonomously acting departments as well.

The basic concepts together with an application scenario are described in section 2. In this section, we also briefly present enabling middleware platforms used for implementation, namely the OSF Distributed Computing Environment (DCE) [20, 30] with the object-oriented DC++ extension [34] and the OMG Common Object Request Broker Architecture (CORBA) [28, 29]. Using these approaches, heterogeneous hardware platforms can be supported. Section 3 details the design and implementation of CodAlf with its architectural components, its workflow specification language, its workflow design approach and its decentralized workflow control support. Some additional features such as object-oriented implementation, security aspects and persistent data support are also discussed. Section 4 presents experiences with this approach and with its applications. We also compare the system with other solutions discussed above in more detail. In particular, we briefly present an alternative solution named BPAFrame ('Business Process Administration Framework') which is based on CORBA. This approach is also a very recent development within our group, initiated as a consequent new analysis and design of a decentralized WfMS. Finally, section 5 concludes with an outlook to future work.

2. CodAlf: foundations and conceptual overview

In this section, we introduce the foundations of our approach using a typical application scenario. We discuss our basic design goals and present OSF DCE and OMG CORBA as two middleware solutions for implementing a distributed workflow management system.

2.1. Application scenario

Figure 1 shows a typical workflow application scenario of the environmental administration area. Before an extensive construction on a building site can begin, an analysis concerning environmental factors has to be performed. It starts with an official inquiry which is handled by an agent. This leads to an analysis of suspicious facts by a dedicated department. Depending on the result, a conditional in-depth analysis concerning potential ground pollution, air pollution, and water pollution is performed by specialized engineers. These tasks can be performed in parallel. Not all tasks have to be performed in any case; their execution depends on special context information of the given project. Each in-depth analysis can consult an existing

database about environmental factors. Typically, additional measurements at the actual location are performed. In special cases, simulations of expected future pollution for special industrial areas are another option of advanced analysis. These three tasks can again be executed in parallel and are mutually optional.

A detailed evaluation of measurements and of simulation results follows. Thereafter, a synchronization of the various tasks is required in order to collect and integrate the results of one category of analysis (air, ground or water). If the results are still too vague, the analysis has to be repeated in more detail, for example by performing additional measurements. Finally, a global synchronization and a final assessment are performed and results are mailed to the client. While the air pollution analysis is shown in more detail in the figure, the other two categories are modelled as complex tasks that are refined separately. Basically, they look similar so the details are not shown here.

From this example, several basic characteristics and requirements of workflows can be derived:

- *Workflow structure:* Workflows typically have a well defined structure and occur repeatedly in practice. The tasks can be executed sequentially (e.g. mailing of results after final assessment) or in parallel (e.g. analysis of database, measurements and simulation). Execution can also be conditional (e.g. simulation of pollution in special cases), and tasks can be optional. Moreover, loops are possible (e.g. for repeating the air pollution analysis). The workflow structure is typically given by a directed *execution graph* (see section 3).
- *Workflow types:* Because many workflows represent collections of routine tasks, they are potentially instantiated many times. Therefore, it is important to make a distinction between workflow types and associated instances. This enables reuse of existing workflow types during instantiation and also during workflow specification: an existing workflow type with its execution graph can be embedded into a new, more complex workflow. Workflow instances have a specific owner (initiator) with general access rights; access of others is usually more restricted.
- *Roles:* The tasks of the given execution graph have to be mapped to actual people, computer systems, tools and applications, i.e. to *execution instances*. For example, the final assessment of results may be performed by an employee using a spreadsheet program. It is important to enable a flexible mapping from tasks to execution instances; for example, several people may be able to perform an air pollution analysis. Therefore, *roles* are introduced: a task is then mapped to a role which can be handled by several execution instances according to a description of the required qualification or the required characteristics.

Based on these assumptions, a formal workflow specification technique and execution model can be introduced as described in section 3. However, additional design goals concerning the implementation of a distributed workflow management system have to be discussed first.

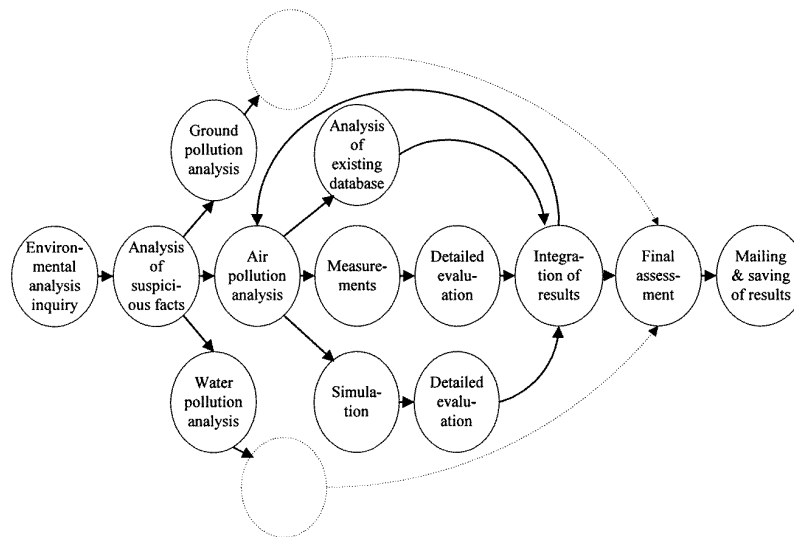


Figure 1. Workflow application scenario.

2.2. Distributed workflow management: design goals

Distributed execution of workflows is required due to the physical distribution of execution instances in typical organizations. As a consequence, the following additional requirements have to be met:

- *Decentralized architecture:* The workflow management system itself should be organized in a decentralized way. That is, it should not depend on a centralized server or database; such components would represent single points of failure and performance bottlenecks. Therefore, the execution instances must be equipped with decentralized runtime components of the workflow management system.
- *Dynamic mapping:* A dynamic mapping of tasks and associated roles onto execution instances must be supported. This way, cost optimizations and means for achieving fault tolerance are enabled. As a basis, tool support such as a trading service [15, 22] must be part of the workflow management system.
- *Control and reliability:* Despite the required decentralization, a logically centralized control of workflow execution must be possible. For example, the initiator of a workflow should always be able to track down the current location and status of workflow execution. Appropriate tools for the management and maintenance of workflows with associated control protocols are necessary, e.g. in the employee's workspace. These tools should be organized in a modular way in order to be combinable and flexible. Control flow shall be organized and persistent.
- *Extendability:* The decentralized workflow management system must be extendable with respect to execution instances, workflow types, and workflow instances. It must be possible to integrate new execution instances and to remove existing instances at runtime. To achieve this, at least a dynamically extensible directory service

is required. In addition, different kinds of application services of execution instances—such as word processing or spreadsheet calculation—should be integrateable via a uniform interface.

- *Scalability:* The workflow management approach must be scalable in terms of the number of execution instances and the number of tasks within a workflow. Therefore, the directory and trading service must be scalable themselves, i.e. they must be able to manage significantly increased amounts of information without a significant loss of performance. Moreover, workflow specification and execution should support hierarchical decomposition in order to enable very large workflows as found in many organizations.
- *Support for heterogeneity:* Finally, workflows are typically executed in heterogeneous organizations. The workflow management system must therefore be portable and must be implemented on top of vendor-neutral, widely available and standardized middleware platforms.

In the following, we discuss the most important middleware platforms, DCE and CORBA, as a basis for implementing our workflow management approach with respect to the given requirements.

2.3. Basic middleware platforms: DCE and CORBA

The basic structure of the OSF Distributed Computing Environment (DCE) [30] and of the OMG Common Object Request Broker Architecture (CORBA) [28, 29] is shown in figure 2. Both platforms are based on existing operating systems (Unix, Windows NT, Windows 95, OS/2 and several host-oriented systems) and on existing transport protocol suites (TCP/IP, UDP/IP and several others).

DCE provides a thread service for concurrent processing, a remote procedure call (RPC) as its general communication mechanism, a directory service

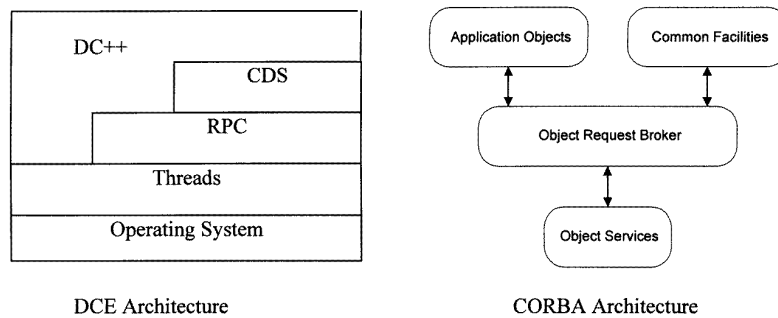


Figure 2. Basic structure of DCE and CORBA.

(CDS—cell directory service and GDS—global directory service) for name and resource management, a security service for authentication, authorization and encryption, and a distributed file system for network-wide file management. Additional products such as Encina and Tuxedo enable distributed transaction processing on top of DCE. Communication and processing is based on the traditional client/server model, using the C programming language as its basis. Although a C++ API is under development, distributed object-oriented features according to [25] are not supported.

Therefore, we developed and implemented DC++, an own distributed object-oriented system on top of DCE [11, 34]. It provides a C++ class hierarchy and runtime system for location independent remote object invocation and object mobility. Objects have globally unique identifiers, and invocations are forwarded to the target location by a forward addressing scheme. Object migrations are performed upon explicit request, for example to co-locate cooperating objects. A migration includes the transfer of all object instance data, the transparent update of inter-object references, and the reinstallation of the object at the target site. Details are found in [34].

As opposed to DCE, CORBA is itself based on a distributed object-oriented approach. It enables communication between application objects at different locations based on its object request broker. In addition to a conventional static invocation interface based on RPC-like stubs, a dynamic invocation interface is also available. It enables the invocation of server objects with dynamic typing—type information of the interfaces need not be available at compile time but can be provided dynamically at runtime. Currently, C, C++, Ada and Smalltalk APIs are available, but the approach is open to other languages. Auxiliary common facilities provide support for managing objects and for integrating existing tools and data, for example via compound document facilities. A set of object services will support naming, concurrency, security, transactions, database integration, versioning and several other aspects [29]. Furthermore, ObjectFacilities are emerging, e.g. for repositories and an interesting approach to a ‘Business Object Model Architecture’ [8] should also be mentioned. The latter will deliver an abstract model not only valid for WfM but also for other aspects in

commercial information systems. A more general and more promising representation of a general business object model is described in [35] as a convergent model. It is not limited but well fitting to ORBs (Object Request Brokers).

However, at the moment, only a limited number of CORBA services is available; security and transactions are not offered as products yet. On the other hand, ORB-provider offers additional tools and interfaces, e.g. for integrating Visual Basic and OLE [14].

Basically, both platforms can serve as an adequate basis for implementing a workflow management system with a decentralized architecture in a heterogeneous environment. Both DCE and CORBA are also commercially available on most systems and have been developed over several years, reaching a level of stability now. The major advantage of DCE with respect to workflow management is that all important services are readily available. Communication between execution instances can be implemented by RPC, instances/servers can be located using the directory service, concurrent invocations within parallel execution paths are enabled by threads, and the security service supports the required security aspects. Optionally, transactions can be employed on top of DCE if required for reliability and recoverability. As discussed, CORBA still lacks security features and transactional facilities, leading to problems in these two areas.

On the other hand, an object-oriented basic architecture can facilitate the implementation of workflow management: workflows can themselves be modelled as (mobile) objects, execution instances can naturally be implemented as objects, too, and inheritance and encapsulation supports reuse and structural clarity. These aspects are not supported by DCE but by CORBA.

Based on these considerations, we decided to take two directions. (1) We designed and implemented a workflow management system named CodAlf on top of DC++, an object-oriented extension of DCE. This work has largely been completed. We therefore describe this approach in detail in the next section. (2) Recently, we also decided to use CORBA as an alternative basis for implementing a workflow management system named BPAFrame. Here, experiences with CodAlf and the basic ideas of the convergent model of business objects [35] influenced the new analysis, design and implementation of

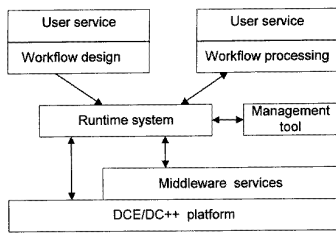


Figure 3. General workflow management system architecture.

BPAFrame. In spite of the discussed limitations, this has been proven to be viable. We discuss our first results and compare them with CodAlf in section 4.

Both approaches are based on a general architecture that is shown in figure 3. A runtime system on top of DCE/DC++ or CORBA offers distributed runtime support for workflow execution. It is based on a middleware platform (DCE/DC++ or CORBA) with additional middleware services and it runs decentralized and autonomously. A management tool enables dynamic control of workflows and of execution instances. The execution instances are equipped with user-level workflow processing software that handles workflow execution and invocation of local applications. Moreover, the workflow design phase is supported by an additional user service for specifying workflow types. All of these aspects are detailed below.

3. Design and implementation of CodAlf

3.1. Workflow design services

Complex workflow applications require a structured description in event lists, graphs, etc, representing structured tasks and parallelism. Such descriptions are a valuable basis for the transformation of often repeated, office applications to distributed systems, for instance based on DCE. Our own Petri-net-based workflow description language has been developed specifically for the mapping of workflow execution onto DCE/DC++. Porting it to other platforms is under consideration, too.

The so called workflow execution graph specifies the various tasks of an overall application and their interaction structure (see figure 4), including a specification of various quality requirements at nodes and edges (transmission paths). The following structural elements describe the variety of execution primitives of workflows:

- sequences;
- parallel edges and alternative edges;
- loops;
- subsequences.

The associated linguistic workflow description consists of two parts. First, the execution graph with general attributes, a data declaration section, an action to be performed during workflow initiation, and a corresponding set of termination actions (e.g. instructions for data handling) are given. Secondly, the detailed description of all task nodes with their required services is required. Furthermore, different kinds of service binding can be specified: static service binding, dynamic service binding via a trader at workflow generation time or stepwise dynamic binding depending on the progress of workflow processing. The definition of an individual task node contains the following information:

- the specification of the service with requested quality parameters
- the kind of server allocation
- the description of the subsequent edges (communication services) with requested QoS-parameters of communication links (e.g. transmission rate)
- control mechanisms (e.g. time-out mechanism, parallel paths or the condition-dependent splitting of the workflow with subsequent synchronization points in both cases)
- resource reservation requests
- determination of the behaviour in the case of errors (currently, rollback to the last checkpoint with retry of the (modified) service request is supported).

A simple example shows a workflow description (in text mode) with the two nodes ‘analyser’ and ‘final assessment’ (see figure 5), it presents an fragment of the workflow execution graph of figure 4.

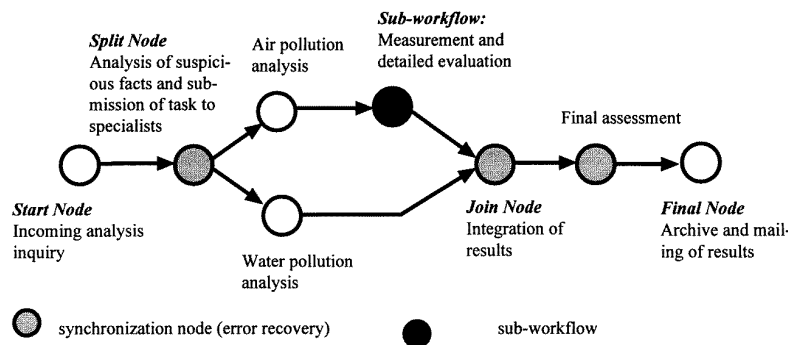


Figure 4. Example of a processing sequence of a distributed application.

```

WORKFLOW environmental_test_frame_A1;
  ATTRIBUTES
    COMPLETION_TIME=8;
    PRIORITY_LEVEL=HIGH;
  DATA
    request_message_type inquiry_request;
    catalogue_type suspicious_facts;
    form_structure_type inquiry_form;
    form_structure_2_type analysis_form;
  INITIATOR
    NOTIFY inquiry_server WITH STATUS procedure started;
  TERMINATION
    COPY inquiry_form TO data_server;
    MAIL inquiry_form TO inquiry_server;

START_NODE analyser;
  SERVICE do_test_suspicious_facts;
    PARAMETERS inquiry_request [IN];
              inquiry_form [OUT];
              analysis_form [OUT]
  NODE_ATTRIBUTES
    time_out = 7;      # seven days
  BINDINGS
    TO_ATTRIBUTES suspicious_facts.air_pollution = to_check;
    SELECT WITH simulation_type AVAILABLE;
  NODE_ACTIONS
    RESERVATION AT_NODE air_pollution_analysis
      RESOURCE employee.name ="Lehmann";
      NOTIFY INITIATOR WITH COMPLETION;
    EXCEPTION
      CAUSE employee.available=false REACTION employee.name="Meier";
  NEXT_NODES
    LINK_TO      air_pollution_analysis;
              water_ pollution_analysis;
  NODE_END
  .....

FINAL_NODE final_assessment;
  SERVICE do_check_results;
    PARAMETERS analysis_form [IN];
              suspicious_facts [IN];
              inquiry_form [IN] [OUT];
  NODE_ACTIONS
    EXCEPTION
      CAUSE analysis_form <> complete REACTION CANCEL;
  NODE_END

```

Figure 5. Example of a workflow description (translated into text mode).

The workflow design phase is supported by an editor with graphical user interface. It is based on the Petri-net-based description of the execution graph, including syntactic and semantic tests. These tests include correctness of the graph structure (directed graph with unique start node and final node; limitation of parallel execution structures to strictly nested structures), correctness of attribute ranges and types, syntactic matching of data types and variables used with additional external definitions, and several other aspects. The tests can also be done interactively during WYSIWYG editing of the workflow graph structure with

our semantic design editor. The graphical description can later be translated into the textual description (and vice versa) after finishing the design.

A workflow generator translates the workflow description (graphical or textual mode) into an internal type description—the input to the runtime system. When a workflow is started, a workflow object is generated and migrated to the first application server as discussed in section 3.3.

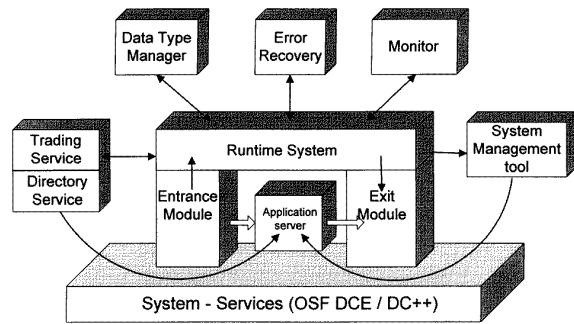


Figure 6. System components.

3.2. System components

Figure 6 shows the overall system structure of CodAlf, detailing the general architecture described above. The runtime system component is installed on any participating node and encloses the application servers. All workflow instances are modelled as mobile objects based on DC++ and move through the distributed system during workflow execution. When a workflow task is to be executed at a specific node, the entrance module receives the workflow object. It is then passed to the runtime system and forwarded again by the exit module after completing the task (see section 3.3). A control module inside the runtime system communicates with the additional tools and middleware services (naming service, trading service, data type manager, error recovery, monitor, and system management tool) in order to control workflow execution.

The directory service maps the names of required execution instances onto addresses of matching servers within the distributed environment. It is possible to consider static attributes describing the server facilities (for example, the level of measurement quality in our environmental application). The service is implemented by using the DCE Cell Directory Service (CDS) with the XDS API [28, 29]. It is augmented with a trading service (see also section 3.5); this service also evaluates dynamically changing attributes of servers to enable an optimized mapping from tasks to servers. The data type manager (see section 3.4) provides type descriptions of workflow data objects at runtime. It is required for dynamic access to arbitrary data objects associated with a workflow. The error recovery component is able to write checkpoints (including the associated data objects) after each task execution and to rollback execution to such a checkpoint. The monitor keeps track of the execution history of a workflow and monitors the executed tasks with the visited servers. In particular, it provides remote control access to a workflow object. This way, the initiator can dynamically query for the current location, execution status, and associated data objects of a workflow. Exception handling can also be controlled remotely, and workflows can be cancelled on explicit demand. A special dialogue and control window is offered for that. The system management tool enables the remote start, shutdown and status control of execution instances in the sense of remote system configuration management. All

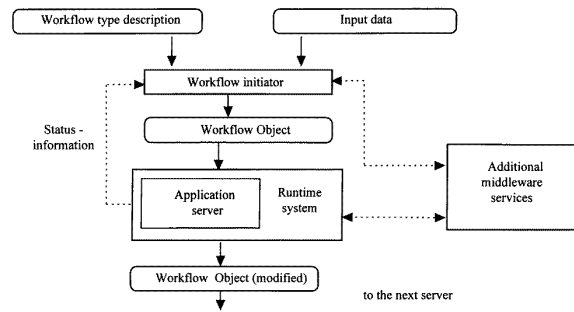


Figure 7. Workflow execution by decentralized runtime system.

of the tools and the runtime system itself use the DC++ and DCE system services.

3.3. Runtime system and core components

The runtime system with its core components performs the actual workflow execution as shown in figure 7. It is realized by active and mobile workflow objects and uses the object migration facility of DC++.

General application servers exist at fixed locations for executing the tasks of the workflow application. A workflow is started by an initiator by selecting an associated type description and by specifying the required input data (for example, an application form for an environmental analysis). The workflow executes while the workflow object moves from application server to server. Every workflow object possesses a copy of the workflow type description. The present state of processing is represented by a state pointer within the execution graph of the type description. After completing a task, status information can be passed back to the initiator via a control protocol of the monitor component. Figure 8 illustrates the core components of the runtime system at a given node in more detail.

Workflow processing at a given node comprises the following major steps and components:

- *The entrance module.* This module receives the workflow object, separates the data and passes it to the application server. The remaining part of the workflow object, i.e. the workflow type description with the execution graph and the status control information, is passed to the control module. For access to workflow data, the data type management module (DTMM) is used as discussed in section 3.4.
- *The control module.* This module prepares the continuation of workflow processing based on the enclosed execution graph and the knowledge of the actual state during data processing by the application server. In particular, conditional execution paths are evaluated, and for all paths to be actually executed, matching servers are selected as execution instances via the trading and directory service.
- *The exit module.* This module receives the data produced or manipulated by the application server,

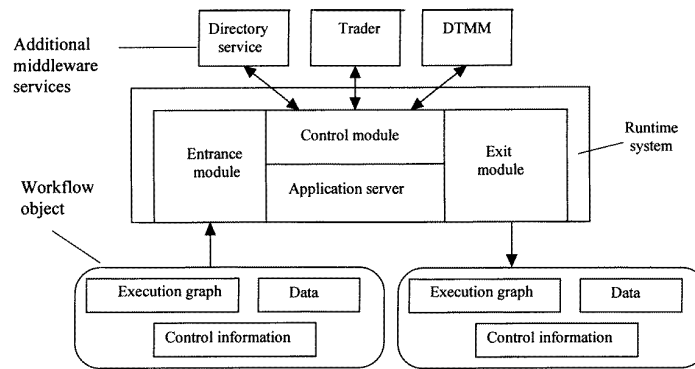


Figure 8. Runtime system, workflow object and application server.

adds them to the modified workflow object and causes the continuation of the workflow in accordance with the execution graph by migrating the workflow object to the next selected server(s). If several execution paths have to be joined before executing a subsequent task (for example, for the final assessment in our example application), a specific synchronization protocol is performed by the exit module: basically, all other involved execution instances have to send a synchronization message to a selected execution instance on the primary execution path that is given within the workflow type definition. Here also resides a error recovery facility (only in synchronization nodes). It realizes a protocol with its predecessor(s) synchronization node (exit module). It is responsible for interaction with the predecessor and with the successor(s). Additionally, it has to realize persistence of control data (the runtime object) and application data via the error recovery module.

The runtime system communicates with the application servers via the application-server interface. Application servers consist of an interface for data exchange and an interface for management purposes. Thus, all internal details of application servers are hidden for CodAlf.

As mentioned above, an additional management system is necessary. It consists of agents on each participating node and allows administrative intervention during workflow execution. The runtime system uses the ability of the management component to start and terminate servers. All application server processes are started initially via the management component. Moreover, the monitor component stores and manages basic events occurring in the distributed system and creates log files of workflow execution.

Workflow processing is described in the faultless case here. For error handling, several different kinds of error semantics are supported by the CodAlf system, for instance for the restart of the application by the initiator or the restart at defined checkpoints (synchronization nodes, see also [22]).

Error recovery is organized between so called synchronization nodes (see figure 4). The protocol works as

follows: synchronization nodes store their processing state (and data) after finishing their task. Then they are informed, if their successor(s) have finished their task(s) and have written to persistent storage. Now, they release their data. Error recovery also contains a protocol for automatic retry (of processing steps). If retry is not allowed or fails, it informs the user via the start node. The facility contains more detail, e.g. for detection and recovery of failures in its own messaging protocol.

Thus, the error handling protocol is also able to fulfil abort and restart orders of the owner of the workflow on demand.

3.4. Data type management

The embedding of the application server into the runtime components requires explicit data management facilities. The runtime system should be applicable for all application servers without changing its source code. On the other hand, it should be possible to pass any data type from the runtime modules to the application server module. We resolve this problem by using a common unique data type which can represent any specific data type using an additional type description. It consists of a byte array with variable length, the byte counter belonging to it and an identifier (UUID) which specifies the real data type. The conversion from a special data type into its common representation and vice versa is carried out by a separate module called data type management module (DTMM).

For each data type used by a workflow, the DTMM generates a type description from a description file with an ASN.1-like syntax and exports it to a type repository. If an application server receives a common data type it passes it to the DTMM. This module obtains the matching type description and decodes it. The communication with the type repository takes place at workflow initiation time; later on, the descriptions are available locally.

The functionality of the DTMM (see figure 9) can be used by almost all components of CodAlf. The workflow initiator can carry out additional semantic tests. The runtime system can evaluate data-based conditions for deciding about alternative execution paths. It can extract any component by element name from a common

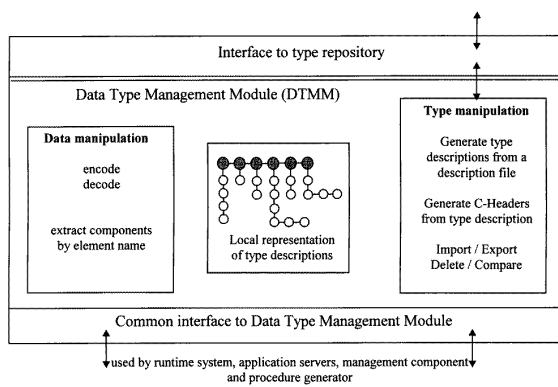


Figure 9. Data type management module.

data type without knowledge of the real data structure implementation. Furthermore, a developer of application servers for CodAlf can generate the necessary C type definitions by using a management interface to the DTMM.

3.5. Use of the trader

The runtime system of CodAlf as well as the initiator of a workflow use naming and trading services. The trading service [15] is realized by our own trader prototype, the X*-Trader, using CDS for static information. The application gets information about matching server objects and their location. Moreover the trader offers a comprehensive server selection method—it includes a variety of server attributes and also dynamically changing attribute values. So, the use of a trader realizes the following qualities of a workflow management system [22]:

- precise server selection according to user demands (based on requirements and constraints, respectively);
- server selection depending on matching attribute values and Boolean expressions of attributes;
- the consideration of system state changes (such as the availability of employees or service providers, or similarly, queue lengths or net loads);
- resource reservations for subsequent steps can be carried out according to predefined resource requirements;
- application development support by service type and implementation repositories and by statistics of successful and unsuccessful service negotiations (feedback for the workflow initiator and the application developer).

We decided to integrate the X*-Trader into our workflow environment based on these considerations. The trader functionality is being used in detail as follows:

- *Static service mapping:* At the beginning, a workflow description is compiled in order to generate an executable form. The trader is used in order to check whether the service of the employee is available and whether the static quality of service requirements can be met at this stage. This is environment dependent and time dependent, of course.

- *Dynamic service mapping:* The control module of each server is responsible for dynamic server mapping. At runtime, it contacts the trader via a specific trader interface and migrates the workflow object in accordance with the associated execution graph. The trader performs static attribute mapping and selects potential server candidates as described above. In addition, it is able to perform an optimized selection of a specific server based on dynamic information.
- *Mid-term optimization:* The knowledge of the trader about successful and failed service binding will be used for a better translation of the service description, optionally. This feature has not been implemented yet.

4. Experiences and comparison with other approaches

After having described the concepts and implementation of CodAlf, we summarize our experiences with this system below. We also compare it with our new, CORBA-based approach and with other systems in more detail.

4.1. Experiences with CodAlf and current limitations

The presented system has been used for designing and implementing several workflow applications. In addition to the environmental administration workflow, a complex image processing workflow and typical office applications (insurance form processing) have been realized. Major experiences are as follows:

- *General goals:* The general goals discussed initially were achieved to a large degree. The architecture is actually decentralized, tasks are dynamically mapped onto servers using the trader, logically centralized control is possible via the monitor tool using location independent access to the workflow object. New servers and workflow instances can be dynamically integrated (workflow types after compilation, too), and heterogeneous platforms are supported based on DCE. Scalability could not be evaluated practically to a significant degree due to the limitations of our environment. However, DCE and DC++ are actually decentralized, and the workflow management system itself also does not necessarily depend on non-scalable, centralized approaches. Though the data type management, the compiler, and the design tools would have to be implemented in a further decentralized way, we do not recognize any basic design limitations that would jeopardize scalability.
- *Genericity and abstraction:* Basically, it makes sense to provide the abstraction of a generic distributed workflow solution. Many example applications have similar characteristics, i.e. well structured tasks, relatively well known execution instances, and a distributed environment. Therefore, a generic solution facilitates the implementation of specific workflows significantly.

- *Separation of concerns:* For such a generic solution, it is very important to separate the general roles of system administration (definition and configuration of execution instances and network structure etc), workflow type definition (specification of workflows and installation within the system), and workflow usage (instantiation and execution of workflows). These roles require different levels of qualification of personnel and should be supported by different tools as found in CodAlf.
- *Object orientation:* The use of an object-oriented middleware (DC++) facilitated the implementation significantly. Resources were designed as a basic class of server nodes. Different kinds of runtime control inherit their behaviour from a common superclass and e.g. synchronization node behaviour. Experiences with former, purely client/server-based solutions confirm this observation; much more effort was necessary there. In particular, object mobility and location independent object invocation enabled a rather straightforward implementation of workflow forwarding and status tracking.
- *Standards:* Finally, the use of standards at the middleware level has proven to be very important. The use of DCE facilitated porting the system from IBM AIX to Digital Unix. Other ports are under consideration. Moreover, a standardized platform also provides a significant level of stability that is not achieved with other *ad hoc* solutions.

On the negative side, the integration of existing tools and applications (such as standard text processing) is rather difficult with the CodAlf approach. A higher-level interface for application integration is required. Moreover, the system is currently not yet available for Microsoft or Macintosh platforms; this would be very important for typical business applications.

Currently, all DCE/DC++ communication is realized without secure RPCs. It will be necessary to develop solutions for the integration of all DCE security levels for the application of CodAlf. A user should determine which security level is required for which kind of workflow in order to establish a high confidentiality of customers, employees and the management of an enterprise. Based on our described system architecture, security features can be integrated without significant conceptual modifications. It is mainly required to use encryption for privacy and integrity of RPC communication, and to authenticate and authorize execution instances, workflow initiators, and management components. Login of all users is already based on DCE-Security-Login so that the basic security control data are already established.

Now, the error recovery facilities are implemented. They allow a consistent rollback of execution paths, also in parallel and conditional paths. It only takes place between synchronization nodes to avoid a huge overhead.

4.2. BPAFrame: CORBA-based workflow management

Our alternative solution is also based on the common conceptual model described in section 2. It also offers

linguistic support for workflow specification and design, a decentralized runtime environment for workflow execution, and tools for administration and management. Its major new features as compared with the CodAlf approach are:

- *Use of CORBA:* Instead of using a specific extension of a procedural client/server platform such as DC++ on top of DCE, distributed object-oriented interaction mechanisms of CORBA are directly employed. All relevant system components are CORBA objects, facilitating the implementation, and providing a higher degree of software stability.
- *Open approach:* The system consists of a loosely coupled collection of tools with well defined CORBA interfaces. It is rather easy to exchange tools and to add new tools, for example for designing workflows, for implementing user interactions, or for dynamic execution control. In this sense, the system represents an open approach.
- *Integration of existing software and data:* Based on a clear separation between client-level applications, object management, and actual data and programs, the integration of legacy software is facilitated. In particular, it is possible to refer to and invoke conventional programs, files and databases from uniform CORBA objects. This way, workflow tasks can be coupled directly with their associated legacy software.
- *Support for document standards:* It is possible to access documents with standard formats directly from CORBA objects. For example, access to Word or Excel documents via Microsoft OLE (Object Linking Environment) is provided. In the near future, it will also be possible to integrate the OpenDoc approach.
- *Naming:* A more comfortable naming concept for objects is supported. All objects (workflows, data objects, execution instances) can be given X.500-compatible names and can be registered and retrieved via an X.500 directory service.

BPAFrame consists of a set of components, class libraries and sub-frameworks, see figure 10. It contains development and runtime support and user interaction services for workflows in open distributed systems.

The business process framework realizes the decentralized control. It uses the agent framework with active objects and all necessary runtime support (i.e. an interpreter for AgentScript—the internal workflow representation language). Together with the organization structure library the user services framework uses the business process framework for all purposes of initialization, supervision and storing of workflows and the active participation of users within workflows. A more abstract application layer comprises front-ends such as monitor, workflow editor and user interaction servers. The latter realize interaction with existing software, e.g. via OLE.

The user services for the user interaction are one of the focal points of the development of BPAFrame. These services contain their own framework within the BPAFrame, called the User Services Framework.

Table 1. Comparison of workflow management systems.

	CodAlf	BPAFrame	Exotica	Meteor2	WorkParty and FlowMark	Mobile	Mentor	Prominand
Decentralized control	yes	yes	yes	both	no	yes	yes	no
Decentralized data-flow	external	yes	yes	both	no	no	no	no
Workflow type/instance separation	yes	yes	yes	yes	yes	yes	yes	yes
Integration of legacy applications	no	yes	yes	yes	yes	yes	yes	yes
Time-out control	yes	no	no	no	no	—	yes	no
Access control	no	no	no	no	yes	—	no	yes
Transactions	no	no	no	yes	yes	yes	yes	yes
Object-oriented architecture	yes	yes	yes	yes	no	yes	yes	no
Standard middle-ware platform	yes	yes	no	yes	no	yes	yes	no

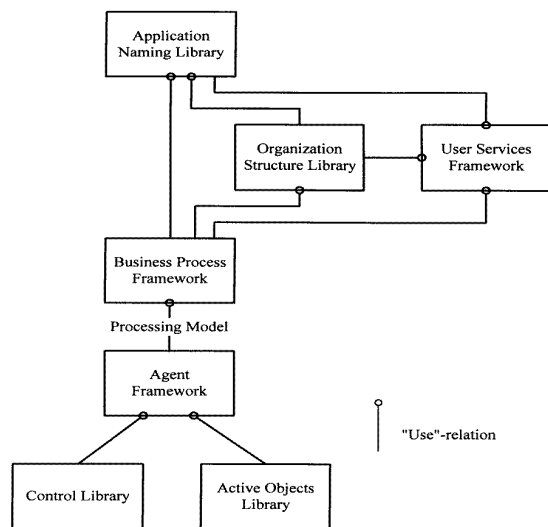


Figure 10. Components of BPAFrame.

Any workflow task is executed by an agent object representing the execution instances. During decentralized workflow execution, the agents access resources (users via user servers, software servers, etc) according to the workflow specification. Here, a business object is a logical representation of data (such as forms and reports) and programs at an intermediate layer. Business objects can reside at remote locations and are accessed dynamically in a uniform way using the CORBA object invocation mechanism. Access to such objects leads to the actual physical activation of the data or program instance at the data layer, for example by loading it from persistent store. Object factories are located on all participating nodes.

The system also supports a specific user interaction protocol as shown in figure 11. A typical interaction occurs when a user manually performs a workflow task that has been assigned to him. Different user interfaces can be supported as long as they conform to the common

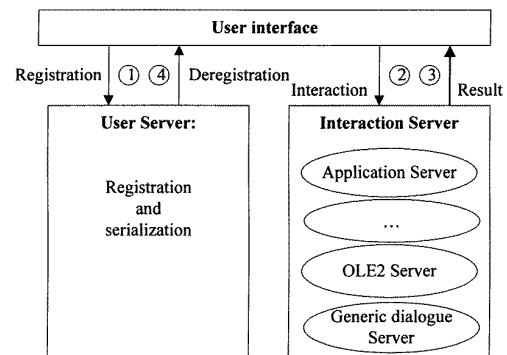


Figure 11. User interaction protocol.

interaction protocol; currently, a Windows-based interface under Windows NT is provided. First, the user registers with a user server component. The user server serializes interactions in order to maintain a proper synchronization. All interactions are then performed by contacting the interaction server via a uniform protocol. Different internal servers are provided, for example an OLE2 server for accessing OLE objects, various application servers, and a generic dialogue server; it supports the interaction via user-defined dialogue boxes. This concept is easily extensible towards new kinds of interaction servers. The interaction server itself is integrated with the workflow management framework via underlying CORBA objects.

The system has been implemented using Windows NT 3.51 with Microsoft Visual C++ and the ORBeline CORBA implementation [31]. Database access from CORBA objects is provided via ODBC. The environmental administration application has been successfully implemented and evaluated with this system in a PC network. In addition to the experiences with CodAlf summarized above, we found that the integration of existing applications is a central issue. The CORBA-based implementation has been completed within only three months by a team of four people

while the DC++-based solution took more than a year due to a much lower level of abstraction of the middleware.

4.3. Comparison with other systems

In section 1, we already mentioned several other systems. Table 1 shows an overall comparison of selected approaches with CodAlf and BPAFrame according to several general criteria. Our own developments as well as Exotica, Meteor2 and Mentor are based on a decentralized, object-oriented architecture. The other solutions which are commercially available use a traditional client/server approach with a centralized server component.

A distinction between workflow types and instances is supported by most approaches, enabling the reuse of existing workflow definitions. BPAFrame as well as the commercial solutions enable the integration of legacy applications. Time-out control of tasks and access control to server components are not supported by all approaches although these are vital requirements. Only some systems support the use and integration of transactions, although this is based on a centralized concept; distributed transactions are not yet supported. CodAlf and BPAFrame use DCE and CORBA, respectively, as standardized middleware platforms while the other approaches are based on lower-level protocols such as TCP/IP or on *ad hoc* communication solutions. Some more approaches are able to use at least CORBA as an additional platform.

Altogether, we come to the conclusion that workflow management in environments of limited scope is already well supported by today's products, but that future applications in larger, widely distributed settings require more advanced, decentralized solutions. Significant efforts are still necessary to integrate the benefits of both areas of computer science and business engineering, i.e. to integrate distribution, transactions, legacy applications, security and a number of additional features discussed in this paper.

5. Conclusions and future work

This paper presented a decentralized, object-oriented approach towards distributed workflow management. We have illustrated that it is possible to separate workflow types from instances, to enable a dynamic mapping of tasks onto execution instances, to forward and synchronize workflows in a completely decentralized way, and to provide adequate tool support. It has also been argued that the implementation of such an approach is further facilitated by the use of CORBA, and that existing applications can more easily be integrated this way. Based on the comparison with other approaches, it has been outlined that the integration of various features still requires additional research and development effort.

Our future work will focus on the implementation and evaluation of additional workflow scenarios, on the extension of the CORBA-based approach concerning the workflow specification technique, and on the integration with other results from our distributed multimedia research [12]. In particular, we are planning to integrate more detailed quality of service specifications into

existing workflow specifications, for example to control communication within image processing workflows.

Acknowledgments

This work has been supported by the Deutsche Forschungsgemeinschaft within SFB 358. We would like to thank all colleagues and students who contributed to the design and implementation of the presented approaches, namely Elke Haubold, Wolfgang König, Torsten Loddert, Steffen Müller, Jörg Schreiter, Birgit Simsch, Kai Sommerfeld, Timo Welker, Gritta Wolf and Jan Zöllner.

References

- [1] Alonso G, Mohan C and Günthör R 1995 Exotica/FMQM: a persistent message based architecture for distributed workflow management *Proc. IFIP Working Conf. on Information Systems for Decentralized Organizations (Trondheim, 1995)*
- [2] Artsy Y 1990 Routing objects on action paths *IEEE Int. Conf. on Distributed Computing Systems (Paris, 1990)* pp 572–79
- [3] Bussler C and Jablonski S 1994 An approach to integrate workflow modeling and organization modeling in an enterprise *Proc. 3rd IEEE Workshop on Enabling Technology: Infrastructure for Collaborative Enterprise (Morgantown, 1994)* pp 81–95
- [4] Bussler C and Jablonski S 1995 Scaleability and extensibility through modularity: architecture of the mobile workflow management system *Proc. 5th Workshop on Information Technology and Systems (Amsterdam, 1995)* pp 98–107
- [5] Bussler C 1995 User mobility in workflow-management-systems *Proc. Telecommunication Information Networking Architecture Conf. (TINA'95) (Melbourne, 1995)*
- [6] Casati F, Ceri S, Pernici B and Pozzi G 1996 Workflow evolution *Proc. 15th Int. Conf. on Conceptual Modelling ER'96 (Cottbus, 1996)* presentation
- [7] Software Ley GmbH 1994 *COSA Documentation*
- [8] Cummings F A 1996 An object model for business applications *EDS contribution to OMG RfP for CORBA facility on Business Objects (1996)*
- [9] IBM 1994 *FlowMark—Modeling Workflows*
- [10] Hammainen H, Eloranta E and Alasuvanto J 1990 Distributed form management *ACM Trans. Inform. Syst.* **8** 50–76
- [11] Heuser L and Schill A 1995 DC++ *Bonn: Int. Thomson Publ. (TAT 15)*
- [12] Hess R, Hutschenreuther T and Schill A 1996 Video communication and media scaling system 'Xnetvideo': design and implementation *Eur. Workshop on Interactive Distributed Multimedia Systems and Services (IDMS'96) (Berlin, 1996)* (Lecture Notes in Computer Science **1045**) (Berlin: Springer) pp 195–210
- [13] Hennessy P, Kreifelts T and Ehrlich U 1992 Distributed work management: activity coordination within the EuroCoOp project *Comput. Commun.* **15** 477–88
- [14] IONA 1995 Orbix—Distributed object technology *Programmer's Guide* (Dublin: IONA Technology)
- [15] ISO/IEC JTC1/SC21/WG7 N963 1994 ODP Trader *Report on the Joint Meeting of ISO/IEC JTC1/SC21/WG7 (Southampton, 1994)*
- [16] Jablonski S 1994 MOBILE: a modular workflow management model and architecture *Proc. Int. Working Conf. on Dynamic Modeling and Information Systems (Nordwijkerhout, 1994)* presentation

- [17] Jablonski S and Stein K 1995 Die Eignung objektorientierter Analysemethoden für das Workflow Management *HMD* **185** 95–115
- [18] Joosten S 1995 Conceptual theory for workflow management support systems *Report* University of Twente
- [19] Krishnakumar N and Sheth A 1994 Specification of workflows with heterogeneous tasks in METEOR *Bellcore Technical Memorandum TM-24198*
- [20] Lockhart H W 1994 *OSF DCE: Guide to Developing Distributed Applications* (New York: McGraw Hill)
- [21] Mohan C, Alonso G, Günthör R, Karnath M and Reinwald B 1995 An overview of the Exotica research project on workflow management systems *Proc. 6th Int. Workshop on High Transaction Systems (Asilomar, 1995)*
- [22] Mittasch Ch, Funke R and König W 1996 Trader supported distributed office applications *Int. Conf. on Distributed Platforms (Dresden, 1996)*. *Distributed Platforms* ed A Schill *et al*, pp 230–44
- [23] Mittasch Ch, Irmscher K, Ziegert T, Müller S and Sommerfeld K 1996 Design and use of BPAFrame—a decentralized CORBA-based WfMS *IFIP World Computer Congress (Canberra, 1996)*. *Advanced IT Tools* ed N Terashima and E Altman (New York: Chapman & Hall) pp 303–10
- [24] Miller J A, Sheth A P, Kochut K J and Wang X 1996 CORBA-based runtime architectures for workflow management systems *J. Database Management (Special Issue on Multidatabases)* **7** 16–27
- [25] Nicol J R, Wilkes C T and Manola F A 1993 Object-orientation in heterogeneous distributed computing systems *IEEE Comput.* **26** 57–67
- [26] Oberweis A 1994 Workflow management in software engineering projects *Proc. 2nd Int. Conf. on Concurrent Engineering and Electronic Design Automation (Bournemouth, 1994)*
- [27] Object Management Group (OMG) 1991 *The Common Object Request Broker Architecture and Specification* Revision 1.1, OMG
- [28] Object Management Group (OMG) 1995 *The Common Object Request Broker: Architecture and Specification* Revision 2.0, OMG
- [29] Object Management Group (OMG) 1995 *CORBA services: Common Object Services Specification*
- [30] Open Software Foundation 1995 *DCE—New Features*
- [31] PostModernComputing 1995 *Orbeline Documentation* Version 1.0
- [32] IABG 1995 *Prominand Documentation*
- [33] Schill A and Gütter D 1994 Extending group communication facilities to support complex distributed office procedures *Int. J. Intell. Cooperative Inform. Syst.* **3** 203–23
- [34] Schill A and Mock M 1993 DC++: distributed object-oriented system support on top of OSF DCE *Distrib. Syst. Engng* **1** 112–25
- [35] Taylor D A 1995 *Business Engineering with Object Technology* (New York: Wiley)
- [36] Weikum G 1993 Extending transaction management to capture more consistency with better performance *Proc. 9th French Database Conf. (Toulouse, 1993)* Invited paper
- [37] Workflow Management Coalition 1996 *WfMC Spec. Terminology and Glossary* Document No WFMC-TC-1011, Issue 2.0
- [38] Workflow Management Coalition 1995 *Work Group 1/3: Interface 1: Process Definition Interchange* WfMC, Draft 2.0
- [39] Siemens-Nixdorf 1995 *WorkParty Documentation*
- [40] Weske M, Vossen G and Bauzer Medeiros C 1996 Scientific workflow management: WASA architecture and applications *Report Fachbericht Angewandte Mathematik und Informatik 03/96-I* Universität Münster
- [41] Wodtke D, Weissenfels J, Weikum G and Kotz Dittrich A 1996 The mentor project: steps towards enterprise-wide workflow management *Proc. Int. Conf. on Data Engineering (New Orleans, 1996)*