Distributed Database Support for Data-Intensive Workflow Applications

S. Bergamaschi[1,2], S. Castano[3], C. Sartori[2], P. Tiberio[1], M. Vincini[1]
e_mail: {tiberio,sonia,vincini}@dsi.unimo.it castano@dsi.unimi.it
csartori@deis.unibo.it
[1] Dip. di Scienze dell'Ingegneria, Univ. di Modena e Reggio Emilia,
[2] Dip. di Elettronica, Informatica e Sistemistica CSITE - CNR, Univ. di Bologna
[3] Dip. di Scienze dell'Informazione, Univ. di Milano,

Abstract. Workflows deal with the automation of business processes in which documents, data and tasks are transferred among participant roles according to a defined set of rules to achieve a common goal. Workflow management systems (WFMS) have mainly focused on implementing the control flow, paying few attention to the data flow. However, data-intensive workflow applications require advanced data handling capabilities, typical of a DBMS, in order to be efficiently supported by the WFMS. In this paper we propose a data replication model, called DOT (Dynamic Ownership Transition), to realize the integration between workflow and database technology to support data-intensive workflow applications. DOT is derived from Distributed DBMS (DDBMS) replication models and allows only one role at a time to have the ownership of data. An automatic mechanism for the dynamic update of the ownership is provided, for a flexible data flow management in enterprise environment. The resulting system offers the functionalities of a WFMS powered by a data handling mechanism typical of Distributed DBMSs. The proposed solution has been implemented in a prototype system integrating ActionWorkflow (Action Technology Inc.) WFMS and Microsoft SQL Server.

1 Introduction

Nowadays, the companies want their data and applications to be opened and distributed closer to the line of business. This means that information systems must support the companies to provide the right information in the right places. WFMSs have emerged as the leading technology to manage the execution of a business process coordinating multiple information resources and several agents spread across the enterprise [2, 8]. These systems are based on a process model representing an enterprise business process as a set of coordinated activities performed by several workflow participants. The emphasis is on the control flow related to the coordination of the activities among the different participants, while data flow management capabilities are poorly considered. Although many WFMSs use a commercial DBMS as back-end server to store workflow and application relevant data, they do not take full advantage of DBMS capabilities in data handling.

In this paper, we propose a new data replication model, called DOT (Dynamic Ownership Transition) and a distributed environment for data-intensive workflow applications, with advanced data handling capabilities of DDBMS. We

exploit the WFMS control capabilities for the control flow and the DDBMS capabilities for the data flow. DOT model allows the management of the data flow in a distributed environment involving several workflow participants. It enforces the "update-anywhere" management, by preventing concurrent update conflicts and by allowing only one owner of the data at a time authorized for update operations. The model supports the automatic and dynamic data ownership change for a flexible data flow management. The proposed model and the associated environment have been implemented on two commercial systems, namely, the ActionWorkflow (Action Technology Inc.) WFMS and its back-end server Microsoft SQL Server. The former, based on a client-server architecture, is strengthened with distributed WFMS functionalities by exploiting the SQL Server Replication Subsystem. The latter is extended with the DOT model.

The paper is organized as follows. In Section 2, we recall the basic concepts of workflow modeling and present an example of data-intensive application. Section 3 introduces the DOT replication model. Section 4 illustrates the proposed system architecture including the virtual distributed WFMS and the DDBMS. Section 5 discusses contributions of our work with respect to related work in the literature, conclusion and future work.

2 Basic workflow modeling concepts and running example

In a WFMS, a business process is usually described in form of a *workflow schema* as a set of activities, properly coordinated to achieve the process goal. Activities (or tasks) are elementary work units. Connections among activities are specified to define the flow structure of the process. Each workflow schema has one start point and several stop points. Connections among activities represent the control flow, that is, the order with which the activities have to be executed in the process. Besides sequence, other connections among activities that can be defined in the flow structure are *split* connections, to initiate concurrent execution, and *join* connections, to synchronize after concurrent execution. Split and join connectors can be composed in order to represent iteration or other complex flow structures, where the same activity can be executed several times within the same workflow execution.

Different types of data are managed by the WFMS for process execution. Besides system and workflow relevant data maintained by the WFMS for a correct execution of each process instance and transition predicate evaluation, *application data* have to be managed, since they are necessary for the accomplishment of process activities. Application data are process specific data (e.g., documents) that are external to the workflow and are managed by the WFMS by invoking external tools, such as for example a DBMS. In the remaining of the paper, we will refer to application data that constitute the input/output of a given activity as *dataset*. Each activity in a workflow schema is then characterized by one or more input datasets and by one or more output dataset(s), depending on how the data flow is defined.

The same workflow schema can be instantiated several times for execution, into the so called process instances. The WFMS executes a process instance by

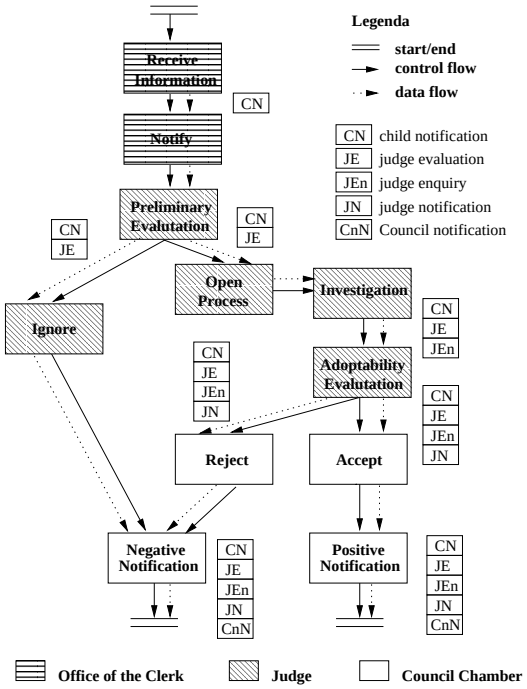

Fig. 1. Example of an Activity Graph related to an Adoption process

scheduling activities according to the flow structure of the workflow schema and by assigning them for execution to a human or automated agent (also called *workflow participant*). For a flexible assignment of activities to workflow participants, commercial WFMSs generally support the definition of an *organization schema*, where workflow participants are grouped into organizational elements, such as, for example, *roles*. In this way, activities in the workflow schema are associated with a role rather than with an individual participant. The workflow participants authorized for activity execution are determined run-time, as an activity is scheduled for execution.

Different workflow models are adopted in commercial WFMSs for workflow schema definition, and currently a standard graphical notation is missing. In the following, we will refer to the workflow model notation proposed by the W*f*MC (Workflow Management Coalition) [2], where a business process is represented as an *activity graph*, where activities are graphically represented by rectangles and activity connections by oriented lines (arrows).

Running example

Let us consider, as an example, a simple data-intensive workflow process in the Public Administration, concerning the process of inserting a child in the national adoption list (according to law 184/83 of the Italian Republic). The activity graph representing this process is shown in Fig. 1. The figure shows the activities, the roles, the control flow, and the data flow characterizing the process.

The process starts when the Juvenile Court is notified of an abandoned child. This notification is registered by the Court's `Office of the Clerk` and then forwarded to the proper `Judge` who starts the legal paper, if he finds sufficient grounds to proceed. In this case, the judge performs preliminary investigations to collect as many information as possible about the child (e.g., existence of relatives who could adopt the child; child's legal and factual status; previous related judgments and laws). At the end of this activities, the case is forwarded to the `Council Chamber` which has to pass judgment about the child's adoptability.

3 The DOT replication model for data-intensive workflow applications

In this section, we first describe basic models of data replication supported by DDBMSs and their implementation, then we propose the DOT replication model specifically conceived for realizing the distributed environment for data-intensive workflow applications.

3.1 Data replication in a DDBMS

The goal of data replication is to provide users in a distributed environment with their local copies of data. These local, updatable data copies can support increased localized processing, reduced network traffic, easy scalability and cheaper approach for distributed processes.

Two basic replication models are generally supported by a DDBMS: synchronous replication and asynchronous replication. The *synchronous replication* (also called *eager* or *tight consistency*) guarantees that all data copies are always synchronized with the original data. This is obtained by using one atomic transaction to update both the original data and its replicated copies. This model reduces performance and availability, since it increases the transaction response time due to extra updates and messages that are necessary for synchronization. The *asynchronous replication* (or *lazy* or *loose consistency*) allows a time interval between the update operation on the original data and the corresponding update operation on its replicated copies. This model maximizes availability and response time, but introduces data integrity problems on concurrent updates.

Such subsystems support both the *synchronous* and the *asynchronous* replication model, but do not manage the concurrent update on the same data from two or more different sites. Consequently, when more than one site tries to update the same record, a concurrency anomaly is detected and the application must manage the exception by a reconciliation algorithm. To avoid these problems the *two phase commit protocol* (2PC), would be useful. 2PC (the *distributed* version [6] and variations of it [4, 5]), is a very simple and elegant protocol that ensures the atomic commitment of distributed transactions. On the other hand, the performance of the distributed commit protocols with respect to the overhead they add is unsatisfactory for the real applications. Thus, at present, concurrent updates on distributed data are left to the responsibility of the application programmer. This choice leads to an increasing difficulty and cost of the application development.

3.2 The DOT Model

To find the most suitable replication model for managing distributed data in a workflow system, we observe that in a workflow application:

- Each activity in the workflow process is characterized by an input set and an output set. The input set contains all the datasets required by the activity to start and accomplish its goal. The output set contains all the datasets resulting from the execution of the activity. During activity execution, each input dataset is manipulated (through read and/or update operations) following the activity procedure rules in the workflow specification.
- Based on the flow structure defined for a given process, it is possible to define the *life-cycle* of each dataset in the process. The life-cycle of a dataset d_i outlines: i) the *states* of the dataset before and after the execution of the activities having d_i in their input/output sets; ii) *state transitions*, determined by the activity execution, according to the flow structure.
- An organizational role is associated with each activity of the workflow, for a flexible assignment of activities to workflow participants.

This way of working suggests the introduction of a replication model to manage updates on replicated data by taking into account the specific characteristics of dataset management in workflow process execution. We call this model DOT (Dynamic Ownership Transition) replication model.

DOT model is a new replication model that implements the "update-anywhere" management, by preventing concurrent update conflicts. To achieve this, the model is based on the concept of *data owner*. The role (and consequently, the workflow participant) designated as the owner of a certain dataset d_i is authorized to insert, delete and update d_i. Other roles are authorized to perform only read operations on d_i. Among all roles, *only one role at a time* can be designated as the owner of a certain dataset (specifically, the role associated with the activity under execution). DOT model provides a dynamic and automatic data ownership change, as the activity execution proceeds, to support all data updates required in the workflow process.

The DOT model is based on the notion of dataset life-cycle. Given a dataset d_i, the life-cycle $L(d_i)$ of d_i is a graph $L(d_i) = \langle N, E \rangle$, where:

- $N = \{n_1, \ldots, n_p\}$ is a set of nodes. Each node $n_j \in N$ represents a state of d_i. In particular, each node is a triple $n_j = (name_j, owner_j, type_j)$, where: $name_j$ is the name of the state in the graph; $owner_j$ is the role that owns the dataset in the state n_j and has the permission to update d_i in the state n_j; $type_j$ denotes the kind of node in $L(d_i)$, with $type_j \in \{$ `start,end,internal`$\}$. Each life-cycle has one `start` node, one or more `end` nodes, while remaining nodes are `internal` nodes.
- $E = \{e_1, \ldots, e_q\}$ is a set of labeled edges. Each edge $e_h \in E$ represents a state transition from a starting state to a destination state of d_i. In particular, each edge is a triple $e_h = (label_h, [source_h], destination_h)$, where: $label_h$ is the label associated with e_h corresponding to the name of an activity of the

workflow schema; $[source_h]$ is the source node of $L(d_i)$ representing the state of d_i before the execution of the activity referred by $label_h$; $destination_h$ is the destination node representing the state of d_i after the execution of the activity referred by $label_h$. The source node $source_h$ of an edge e_h can be missing (denoted by the symbol [ ]); in this case, the destination node $destination_h$ is the start node of $L(d_i)$.

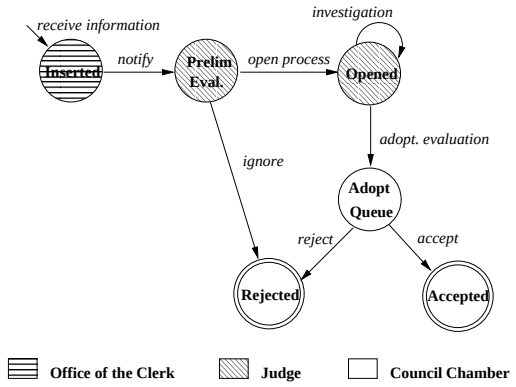

Fig. 2. Data State Transition Graph for the `Child legal document` dataset

According to this definition, the life-cycle of a dataset is represented as a graph, called *Data State Transition Graph (DSTG)*. For example, with reference to the Adoption workflow process of Fig. 1, the DSTG for the `Child legal document` dataset is shown in Fig. 2, where circles represent nodes and arrows state transitions. The `Inserted` state is the start state, while the states having a double circle border (i.e., `Rejected` and `Accepted`) are the final states. Arrow labels correspond to the names of activities in the workflow schema of Fig. 1. The owner of a state (i.e., the role responsible of the activity referred by the arrow entering the state) is graphically shown with a pattern inside the corresponding circle.

There is a close relationship between the two graphs of Fig. 1 and Fig. 2. The former represents the workflow process schema, by showing the activities to be undertaken to accomplish it and their connections. The latter represents the changes in state that a given dataset undergoes as a consequence of the activities of Fig. 1 as the workflow execution proceeds, therefore the edge labels are given by activity names of Fig. 1.

Let us examine the `Opened` state. When the `Child legal document` is in this state, the `Judge` has already been notified about the child, and agreed to open a case. He is, therefore, investigating to collect more information about child's position. The `investigation` transition exactly represents this activity. The state is exited only when the `Judge` closes the preliminary investigation (`adoption evaluation` transition) forwarding the `Child legal document` to the `Council Chamber`. Notice that during the period in which the `Child legal`

`document` is in judge's owned states, only the `Judge` himself has the authorization to update it. In the meantime, other roles can perform only read operations on the `Child legal document` (specific triggers prevent every possible conflict, as discussed below).

By constructing the DSTG graphs for all datasets and all roles involved in a workflow schema, we can apply the DOT model and manage all data flows involved in the workflow process execution. The strict relationship existing between the activities of the flow structure of a workflow schema and the transitions in the DSTG graph is the basis to allow the integration of the control and data flows management in a distributed system architecture integrating database and workflow technology, based on the DOT model.

3.3 Dynamic Ownership Transition

At the implementation level, the model is enforced by means of triggers. Triggers are defined to check that, on any requested insert, update, or delete operation on a given dataset, the requesting workflow participant (i.e., the agent actually executing the activity from within the operation is requested) has the right authorization for the requested operation. Moreover, triggers are defined to enforce the dynamic ownership change in the system.

The data access permission and the dynamic ownership change process are based on the *state* of a dataset. A dataset d_i is implemented in a relational DDBMS as a set of fragments of tables. Thus, d_i state is represented by adding a *state* column to each table of d_i and by three DOT support tables (see below). Suppose to have a simple dataset containing information about child; the *state* column is added to describe its actual state in the Adoption Process. A possible instance of this child dataset is shown in Figure 3.

`Child_Table`

name	**birthdate**	...	**state**
...	...	...	...
Joe	02/07/94		Rejected
Mary	12/06/95		Opened
Paul	05/06/91		Eval
Mark	09/01/97		Opened
...	...	...	...

Fig. 3. Example of child dataset

The following DOT support tables are defined to store the DSTG of a dataset d_i in the distributed database:

`State_Table` : specifies, for each possible state of d_i, the owner role authorized to modify d_i, a flag specifying the kind of state (i.e., start, internal, end state);

`State_Trans_Table` : describes the possible state transitions for d_i, by specifying the involved activity, as well as the source and destination state;

`Role_Server_Table` : specifies the allocation of a role to a specific server in the distributed environment, and has schema `(role, server_name)`.

Fig. 4 shows the tables (automatically generated by the DOT Builder software module, see Section 4.2) for the DSTG of Fig.2.

`State_Table`

state_name	owner	type
Inserted	Clerk	start
Eval	Judge	internal
Opened	Judge	internal
AdoptQueue	Council	internal
Rejected	Council	end
Accepted	Council	end

`Role_Server_Table`

role	server_name
Clerk	Clerk_Server
Judge	Judge_Server
Council	Council_Server

`State_Trans_Table`

label	source	destination
Receive Inform.	NULL	Inserted
Notify	Inserted	Eval
Open Process	Eval	Opened
Ignore	Eval	Rejected
Investigation	Opened	Opened
Adopt. Evaluation	Opened	AdoptQueue
Reject	AdoptQueue	Rejected
Accept	AdoptQueue	Accepted

Fig. 4. Example of support DOT Tables for the DSTG of Fig.2

Suppose that a `Judge` agent, after investigating and evaluating on Mary's adoption, decides to send the child dataset to the `Council Chamber` for the final sentence. This involves an update for the state of `Mary` dataset from the `Opened` value to the `AdoptQueue` value. Based on the information stored in the support tables, the system verifies that the `Judge` is the owner of the `Mary` dataset (`State_Table`), and that the transition from the `Opened` to the `AdoptQueue` state is admissible (`State_Trans_Table`). The update request is successfully executed and the `Mary` dataset is updated to the `AdoptQueue` state. This also involves the ownership change for `Mary` dataset from the `Judge` role to the `Council` role.

In general, for each request regarding an insert, update or delete operation on a dataset d_i from within an executing activity, the following actions are performed:

1. storing of the state before, say S_1, and the state after, say S_2, for the operation requested on d_i;
2. checking that the role requesting the operation is the owner of S_1 and that the transition from S_1 to S_2 is admissible, based on the information reported in the `State_Trans_Table`;

3. performing the requested operation if the check is positive;
4. changing the ownership of d_i, if the owner of S_2 in the `State_Table` is different from the owner of S_1.

The above actions are implemented by triggers, which are automatically generated by the DOT Builder software module (see Section 4.2).

4 A distributed workflow architecture based on DOT

In this section, we describe the distributed workflow architecture of the proposed system that combines ActionWorkflow for the management of the control flow and SQL Server with its Replication Subsystem for the management of the data flow.

4.1 Making ActionWorkflow a virtual distributed WFMS

ActionWorkflow is a client-server WFMS working in connection with a back-end server managing its databases.

Each ActionWorkflow installation consists of two subsystems: Business Process Builder and Process Manager (see [9]). The former is a stand-alone program that enables users to draw the Activity Graph (here called *Business Process Map*, BPM), and to compile it; the latter must be installed on the server and manages process loading and execution. ActionWorkflow works with two databases, called *aws* and *awsarch*. All the operations made on process instances are, on a lower level, simple read/write/update operations on tables of the *aws* database; i.e. it is the real implementation of an executable process. *Awsarch* archives data of completed actions.
In order to make ActionWorkflow work as a "virtual distributed" workflow system we need to a) install an ActionWorkflow Process Manager on each site of the network and b) configure a replication schema for the *aws* databases through the DDBMS Replication Subsystem.

In this way all the servers can virtually operate on the same process instance, even if there is a replicated process instance for each site server.

4.2 The distributed system architecture

The system architecture we propose is shown in Fig. 5. The DOT components interact with the ActionWorkflow components both at configuration time and run time.
Configuration phase The goal of the configuration phase is to define the workflow schema (i.e., the control and data flow and involved activities) and to set up the data distribution.

- **BP Builder**: it allows the definition of the workflow maps[4], workflow loops and links in the definition database, and provides an automatic checking of the map based on the ActionWorkflow System rules. With reference to our example, the Activity graph of Figure 1 is drawn with this component.

[4] The ActionWorkflow Business Process Builder uses a graphical notation slightly different from the one used by the WfMC; to be general, we used the WfMC notation.

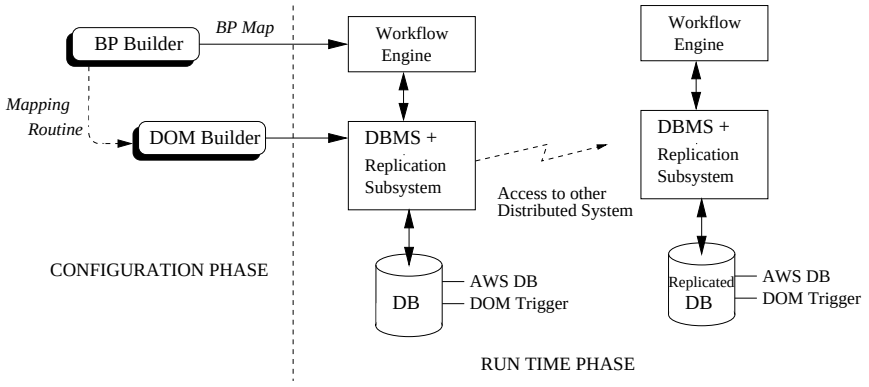

Fig. 5. Proposed system architecture

– **DOT Builder**: it is the prototype tool we developed to support the definition of the DSTGs (by a graphical interface). DOT automatically generates the script files containing the triggers and the checks to implement both the dynamic ownership behavior and the data distribution on the DBMS. In particular the data distribution is obtained by defining a star network (see Fig. 6 for an example network), where each node is a server site that maintains the information about the roles authorized to execute the operations at this site and a "virtual" *master site* is created, which coordinates the broadcast replication to the peripheral nodes. To prevent concurrent updates on workflow instance and data within the distributed network, our approach limits the execution grant of a role to a single site server, while in a site can coexist many roles. This constraint does not result in a loss of generality as a BP Map role is often physically tied to a single department (server site). Note that, by using DOT Builder, we simply introduce the list of the nodes with their associated roles, a node as a master site and the star network functionalities for data distribution and automatically generated. The replicated transactions move in both directions between the local server sites and the master site (no transaction can be moved between two local servers). The two types of publications are called PUSH and PULL respectively: i) the PUSH publication, at a local server site, sends the data modified by the owner to the master server; ii) the PULL publications, at the master site (one for each server site), contain the data to be distributed at each server site.

System set up This section describes the steps required to set up the whole distributed environment.

1. *Install ActionWorkflow and SQL Server at each network site.*
   In order to provide the distributed functionalities, each single site requires both the workflow engine (ActionWorkflow Process Manager) and SQL Server to manage locally the control and data flow of the BP Map instances.
2. *Install the BP Map into the Master Server.*
   In order to provide an executable BP Map to the workflow engine, it is neces-

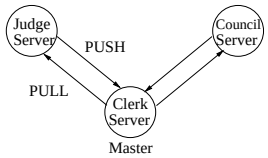

Fig. 6. Adoption Child network topology

sary to import the proper BP Map into the Master server's ActionWorkflow Process Manager.

3. *DOT Tables and Trigger implementation.*
   This operation, denoted in Fig. 5 as DOT Trigger, consists in adding to the *aws database* of the master server, the following tables, triggers and fields:
   - DOT Tables: `State_Table`, `State_Trans_Table` and `Role_Server_Table` described in Section 3.3;
   - Check Triggers: the set of triggers which manage transitions according to the DSTG of the activity;
   - DOT Fields. A new field is added to the application data tables in order to implement the dynamic ownership mechanism (state attribute).
4. *Synchronize all workflow engines.*
   In order to provide a unique BP Map description and its unique corresponding DSTG for all the engines within the network, it is necessary to replicate the Master server's *aws* tables to all involved sites. Operating this way, the process is installed and then executed virtually as unique, and can be managed by the resulting system in a distributed way. Notice that the *aws* database replica configuration is automatically performed by the system by using the script files generated by DOT Builder.

Run time phase During the run time phase the distributed workflow engine manages the business process instances and uses the *aws* database to determine which acts have been completed and which acts can be performed. It includes a scheduler process and a transaction manager to execute workflow instances. Thanks to the set-up operation 3 and 4, its behavior is distributed and supports dynamic ownership change.

5 Related work Concluding remarks

The approach of distributing the data in a workflow environment was proposed in [7] and in [1]. In [1] the main idea for the integration of the control and data flow consists to use the replication capabilities of Lotus Notes [3] to define a loosely synchronized replicated database. A similar approach has been adopted in [7], where the data flow is realized on top of a distributed relational database system prototype.

In both proposals, and also in our approach, the main goal is that any data required for the execution of a workflow activity be writable/updatable at the execution site, and readable at the other sites. The papers discuss also the transactional implementation, using a message communication protocol among the nodes to ensure the atomicity of an activity. Our approach, instead, uses and configures the distributed replication protocol management offered by the DDBMS. Furthermore, the above approaches support simple data types (that are integer,

booleans, string), while the definition of the dataset permits, in DOT, to manage complex data implemented by relational tables.

In this paper, we have presented a novel integrated architecture to manage the control flow and the data flow in distributed, data-intensive workflow applications. The architecture relies on the DOT replication model, based on the concepts of data owner and dynamic ownership change, to manage distributed updates to the application data used by the workflow application.

Original aspects of the proposed distributed architecture are related to the presence of local workflow engines in each site connected to the external DDBMS Replication Subsystem, to provide distribution around the network. A prototype of the system has been realized at the DSI-University of Modena, using ActionWorkflow, SQL Server and Visual C++ for the DOT Kernel development.

Future work will be devoted to the implementation of a configuration GUI and mapping rules to help the designer in the integration between control and data flow to improve the integration between DOT Builder and the Action Workflow Business Process Builder.

References

1. G. Alonso, B. Reinwald and C. Mohan, "Distributed Data Management in Workflow Environments". *Research Issues in Data Engineering (RIDE), Proc. of the Int. Work. in Birmingham, UK*, pag. 82-90, IEEE Computer Society Press, 1997.
2. D. Hollingsworth, "Workflow Management Coalition: The workflow reference model", *Document WFMC-TC-1003, Workflow Management Coalition, Nov. 1994*, accessible via http://www.aiim.org/wfmc/.
3. L. Kawell, S. Beckhradt, T. Halvorsen, R. Ozzie and I. Greif, "Replicated document management in a group communicating system", in *Proc. of the Conf. on Computer Supported Cooperative Work, CSCW(Portland, Oregon)*, 1988.
4. C. Mohan and B. Lindsay, "Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions", *in 2nd SIGACT-SIGMOD Symp. on Principles of Distributed Computing*, pag. 76-88, ACM, 1983.
5. C. Mohan, B. Lindsay and R. Obermarck, "Transaction Management in the R* Distributed Database Management System", *ACM Trans. Database System*, pag. 378-396, N. 11, S. 4, ACM, 1986.
6. M. T. Ozsu and P. Valduriez, "Principles of Distributed Database Systems", *Prentice Hall International Editions*, New Jersey, 1991.
7. B. Reinwald and H. Wedekind, "Automation of Control and Data flow in Distributed Application System" in *Proc. of DEXA, in Valencia, Spain*, pag. 475-481, Berlin 1992, Springer-Verlang.
8. H. Stark and L. Lachal, "Ovum Evaluates Workflow", *Ovum Evaluates*, Ovum Ltd., London, 1995.
9. Action Technology Inc., "ActionWorkflow Enterprise Series 3.0: ActionWorkflow Guide", 1993-1996.

This article was processed using the LaTeX macro package with LLNCS style