Component-Based Development of Dynamic Workflow Systems Using the Coordination Paradigm

George A. Papadopoulos and George Fakas

Department of Computer Science
University of Cyprus
75 Kallipoleos Street, P.O. Box 20537, CY-1678, Nicosia, CYPRUS
{george,fakas}@cs.ucy.ac.cy

Abstract. We argue for the need to use control-based, event-driven and state-defined coordination models and associated languages in modelling and automating business processes (workflows). We propose a two-level architecture of a hierarchical workflow management system modelled and developed in such a state-of-the-art coordination language. The main advantage of a hierarchical, coordination-based architecture is that individual workflow entities can be easily replaced with others, without disrupting the overall workflow process. Each individual workflow entity exhibits a certain degree of flexibility and autonomy. This makes possible the construction of workflow systems that bring further improvements to process automation and dynamic management, such as dynamic (re-) allocation of activities to actors, reusability of coordination (collaboration) patterns, etc. A case study is presented to demonstrate the use of our approach.

Keywords: Component-Based Systems; Coordination Models and Languages; Workflow Systems; Dynamic (Re-) Configurable Systems; Collaborative Environments.

1 Introduction

Workflow management is concerned with the coordination of the work undertaken by a number of parties. It is usually applied in situations where processes are carried out by many people, possibly distributed over different locations. A workflow application automates the sequence of actions and activities used to run the processes. Such an ensemble of cooperative distributed business processes requires coordination among a set of heterogeneous, asynchronous, and distributed activities according to given specifications.

Therefore, it is not surprising that a number of researchers have proposed workflow models, where the notion of coordination plays a central role in the functionality of their frameworks. Typical examples are DCWPL ([7]), a coordination language for collaborative applications, ML-DEWS ([8]), a modelling language to support dynamic evolution within workflow systems, Endeavors ([10]), a workflow support system for exceptions and dynamic evolution, OPENflow ([20]), a CORBA-based workflow environment, and the framework proposed in [11]. A notable

V. Malyshkin (Ed.): PaCT 2003, LNCS 2763, pp. 304–315, 2003.

common denominator in all these proposals is the fact that they take seriously issues of dynamic evolution and reconfiguration. Interestingly, another notable common denominator is the fact that the line of research they pursue seems to be quite independent from similar research pursued in Component-Based Software Engineering (CBSE), particularly within the subfield of coordination. It is precisely this relationship between coordination in CBSE and workflow systems that we explore in this paper.

More to the point, we have seen a proliferation of the so-called *coordination models* and associated programming languages ([17]). Coordination programming provides a new perspective in constructing software programs. Instead of developing a software program from scratch, the coordination model allows the gluing together of existing components. Whereas in ordinary programming languages a programmer describes individual computing components, in a coordination language the programmer describes interrelationships between collaborating but otherwise independent components. These components may even be written in different programming languages or run on heterogeneous architectures.

Coordination as a science of its own whose role goes beyond software composition, has also been proposed ([11, 12]). However, using the notion of coordination models and languages in modelling workflows, the so-called *coordination language-based approach to groupware construction* ([6]), is a rather recent area of research. Using such a coordination model and language has some clear advantages, i.e. work can be decomposed into smaller steps which can be assigned to and performed by various people and tools, execution of steps can be coordinated (e.g. in time), and coordination patterns that have proved successful for some specific scenario can be reused in other similar situations. Furthermore, this approach offers inherent support for reuse, encapsulation and openness, distribution and heterogeneous execution. Finally, the coordination model offers a concrete modelling framework coupled with a real language in which we can effectively compose *executable specifications* of our coordination patterns.

The rest of the paper is organised as follows. In the next section we present a specific coordination model and associated language, namely IWIM and Manifold. This is followed by the presentation of a hierarchical workflow coordination architecture, where we show how this can be used as the main paradigm for modelling workflow activities. We then validate the proposed architecture by using a case study. We end with some conclusions and description of related and further work.

2 The Coordination Model IWIM and the Manifold Language

In this section we describe a framework for modelling workflows in the coordination language Manifold (and its underlying coordination model IWIM). As will be explained in the next section, Manifold plays the role of the execution environment for the workflow model presented there. The IWIM model ([3]) belongs to the class of the so-called control-oriented or event-driven coordination models. It features a hierarchy of processes, playing the role of either computational processes or coordinator processes, the former group performing collectively some computational

activity in a manner prescribed by the latter group. Both types of processes are treated by the model as black boxes, without any knowledge as to the constituent parts of each process or what precisely it does. Processes communicate by means of well-defined input-output interfaces connected together by means of streams.

Manifold is a direct realisation of IWIM. In Manifold there exist two different types of entities: *managers* (or *coordinators*) and *workers*. A manager is responsible for setting up and taking care of the communication needs of the group of worker processes it controls (non-exclusively). A worker on the other hand is completely unaware of who (if anyone) needs the results it computes or from where it itself receives the data to process. Manifold possess the following characteristics:

- *Processes*. A process is a *black box* with well defined *ports* of connection throughwhich it exchanges *units* of information with the rest of the world.
- *Ports*. These are named openings in the boundary walls of a process through which units of information are exchanged using standard I/O type primitives.
- *Streams*. These are the means by which interconnections between the ports of processes are realised.
- *Events*. Events are broadcast by their sources in the environment, yielding *event occurrences*.

Activity in a Manifold configuration is *event driven*. A coordinator process waits to observe an occurrence of some specific event (usually raised by a worker process it coordinates) which triggers it to enter a certain *state* and perform some actions. These actions typically consist of setting up or breaking off connections of ports and channels. It then remains in that state until it observes the occurrence of some other event which causes the *preemption* of the current state in favour of a new one corresponding to that event. Once an event has been raised, its source generally continues with its activities, while the event occurrence propagates through the environment independently and is observed (if at all) by the other processes according to each observer's own sense of priorities.

More information on IWIM and Manifold can be found in [3, 5, 15, 16, 17] and another paper by the first author in this proceedings volume.

3 A Hierarchical Workflow Coordination Architecture

The motivation behind our approach lies in the observation made in [10] that „traditional approaches to handling [problems related to the dynamic evolution of workflow systems] have fallen short, providing little support for change, particularly once the process has begun execution". Intelligent process management is a key requirement for workflow tools. This is catered for in our approach as agents of the underlying coordination model are able to manage themselves. In particular, workflow processes are modelled and developed in a number of predefined interrelated entities which together form a meta-model i.e. process, activity, role, and actor. We propose a hierarchical architecture where the individual workflow entities can be easily replaced with others, without disrupting the overall workflow process.

Each individual workflow entity exhibits a certain degree of flexibility and autonomy. This makes possible the construction of workflow systems that bring further improvements in process automation and dynamic management, for example dynamic (re-) allocation of activities to actors. In that respect, we advocate the approach proposed in [8] which involves a two-level hierarchy: the upper level is the specification environment which serves to define procedures and activities, whereas the lower level is the execution environment which assists in coordinating and performing those procedures and activities. In this section we describe the top level (itself consisting of a number of sublayers), whereas in section 4 we show how it can be mapped to the lower (execution) level, realized by the coordination language Manifold. Figure 1 below visualises the layered co-ordination workflow architecture. Agents of each layer utilise (trigger) agents from the layer below. The hierarchical nature of the architecture allows flexible workflow systems to be designed in a modular way.

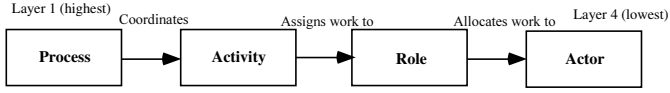

Fig. 1. A Hierarchical Workflow Management Architecture

3.1 Process

A process is a collection of coordinated activities that have explicit and/or implicit relationships among themselves in support of a specific process objective. A process is responsible for coordinating the execution of activities. Its main functionality therefore is to manage, assist, monitor and route the workflow. Process objects are able to manage the execution of the workflow:

- **Via alerting using deadlines.** A deadline is assigned for every activity. If the activity is not completed before the deadline, the process is responsible to send an alert message to the activity.
- **By prioritising.** Every activity is characterised by a priority level relative to other activities. This knowledge is used by the Process object for more efficient task allocation and scheduling.
- **By real-time monitoring.** The process keeps track of parameters related to its execution such as Total Running Time, Current Activity and its status (Waiting Time, Deadline, Role and Actor Selected), etc. This information is useful to trace any bottlenecks in the process.
- **By estimating the time and resources required for execution.** The process is capable of estimating the total duration of the execution and the resources required. It achieves this by interrogating the activity objects ,which in turn may query role objects and so on.

  The following table summarizes the events that trigger a process and its states.

Process	
Event	**State**
Start process	Triggers the process activities. Process is responsible for coordinating activities and the sequence and rules of activities execution.
Process administrator examines process status	Process reports current state; i.e. Total Running Time, Current Activity and its status (Waiting Time, Deadline, Role and Actor Selected), etc.

3.2 Activity

An activity is a single step within a process definition that contributes to the achievement of the objective. It represents the smallest grain of abstracted work that can be defined within the workflow management process. Every activity is related to a role (which is going to perform the work) and to in/out data. An activity instance monitors the execution of the work over time by maintaining information about the activity such as: deadline, priority, estimated waiting time or execution time. The following table summarizes the events that trigger the activities and their states.

Activity	
Event	**State**
Process triggers activity	Receives in in-tray activity input and then assigns the work to the relevant role; then waits until activity deadline expires or executed.
Actor executes activity	Finished, put output in out-tray.
Activity deadline expires	Every activity is associated with a deadline; when this expires the activity asks the corresponding role to examine the actors workload and take the appropriate actions.

3.3 Role

It is important to define roles independently of the actors who carry out the activities, as this enhances the flexibility of the system. Roles assign activities to actors. If an actor is unavailable (e.g. an employee is ill) then somebody else is chosen to carry out the activity. Role objects have the following features and responsibilities:

Allocation of activities to actors. It is the role's responsibility to allocate activities to actors. Its aim is to make an optimized allocation of work which is dynamic by taking into account parameters such as:

- **The actor's level of experience.** Actors have different levels of experience (novice, expert or guru) in performing an activity. Typically, an activity will be allocated to actors with the highest level of expertise available.

- **The actor's workload.** Actors with a heavy workload are less preferable when activities are allocated by roles.
- **Allocation by role-base reference.** In the case of process loops, roles can allocate iterated activities either to the same actor or to a different one.

Report Actors Overload. The role examines the actors' workload and if none of the actors are able to execute the activity before its deadline because they are overloaded, then the role notifies the activity.

If the role discovers an actor that will not be able to execute any of the activities allocated to it before their deadlines then the role might try to *reallocate* the work. For reallocation of work, the same criteria are used (i.e. taking into account the actor's level of experience, workload, use of role-based references, etc.).

The following table summarizes the events that trigger the roles and their states.

Role	
Event	**State**
Activity assigns work or deadline expires	Role checks its actors' workloads. If none of the actors is able to execute the current activity before its deadline because they are overloaded then the role deals with overload.
Role assigns work to actor	Receives in in-tray activity input and then assigns the work (and associated input) to an actor according to some criteria: actor's level of experience, actor's workload and role-based reference, and then waits until work is executed or reassigned to another actor.
Deadline expires and actors are not overloaded	The role is checking up whether it is preferable to reassign the activity to a different actor less busy to perform it or just alert the user responsible for it.
Role reassigns work to a different actor	The role reallocates those activities to other actors. Reallocation of work considers the same criteria as initial allocation of work does. When finished, put output in out-tray.
Role alerts actor	The role alerts the actor responsible for performing the activity.
Actors are overloaded	Deal with actors' overload by either extending the activity's deadline, allocating more actors to the process, or changing the activities' priorities

3.4 Actor

An actor can be either a person or piece of machinery (software application, etc.). Actors can perform and are responsible for activities. Actor workflow objects have the capability to schedule their activities. Activity scheduling is done using policies such as the earliest due job is done first, the shortest job is done first, etc.

The following table summarizes the events that trigger the actors and their states.

Actor	
Event	**State**
Role assigns a work	Receives work in in-tray
Actor schedules his work	The way the actor schedules his work i.e.: FIFO, Shorter First and etc.
Executes work	Executes work and puts output in out-tray
Reports overload	The actor can manually report overload and then the corresponding role will try to solve it

4 A Case Study

The expenses claim process has been used to validate our approach. It is a very common administrative process where an employee is claiming his/her expenses back from the company. The employee fills in a claim form and then sends it to an authorized person for approval. An authorized person could be the head of the department's secretary. In case where the amount claimed is over 1,000 pounds, it must be approved by the head of the department. If the authorized person does not approve the employee's claim, then (s)he sends a rejection message back to the employee; otherwise (s)he sends a message to the company's cashier office to issue a cheque. Finally, the cashier issues and sends a cheque to the employee.

The following table shows how the above scenario is modelled in IWIM. The Expenses Claim Process is a manager entity and the rest are worker ones.

Expenses Claim Workflow Process		
Activity	**Role**	**Actor**
Claim (employee)	Employee	Actor AP1
Approve (Authorized Person)	Authorized Person	Actor HP1
Approve (Head of Dept)	Head of Dept	Actor C1
Pay (Cashiers)	Cashiers	Actor C2

The following coding shows the process logic that contains the process activities and is activated when the process starts. We use a user-friendly pseudo-Manifold coding which is more readable and dispenses us with the need to provide a detailed description of how we program in this language, something we would rather avoid due to lack of space. This pseudo-code however is directly translatable to the language used by the Manifold compiler. Every time a user wishes to start a claim process, an instance of the process and its activities are constructed. When the user finishes with the `putClaim()` activity then the next activity will be called. Assuming that the claim is less than 1000 pounds then the `approve()` activity by the authorized person is called. Then `authorisedPerson` role is assigning the work to an actor. The role, before assigning the work, examines all actors workload (i.e. checks whether any actor can perform the activity before its deadline). If all the actors of a role are overloaded and are not able to perform extra work, then the role has to deal with the actors overload (`DealWithActorsOverload` state) and solve

the overload problem either by extending the activity's deadline or by allocating more workers to the process; otherwise, the role assigns the work to an actor. The activity is in a waiting state until either the actor assigned the work performs it or the activity's deadline expires. If the activity deadline expires before the actor performs it, then the role examines again whether to reassign the work to a different actor or just send an alert message.

Eventually, when the activity is executed, the process proceeds to the next activity, i.e. cashier issues `Cheque()` (if authorised person approves payment). Again, all these activity actions are taken dynamically to manage the process execution.

```
Manifold Process (port in, port out).
Manifold Activity (port in,port in, port out, port
out).
Manifold Role (port in, port in, port out, port out).
Manifold Actors (port in, port out).
Manifold ClaimForm, ApproveForm, PaySlip.

Manifold main
{
 event processMonitoring, assignActivityToRole,
deadlineExpires.
 auto process ClaimExpenses is Process.
 auto process startClaim, ApproveAuthPer,
              ApproveHeadDep, Pay is Activity.
 auto process Employee, AuthorisedPerson,
              HeadOfDept, Cashiers is Role.
 auto process ActorAP1, ActorHP1, ActorC1, ActorC2 is
Actor.

 begin: (ClaimExpenses -> ApproveHeadDep ->
AuthorisedPerson
        -> ActorAP1, ClaimExpenses -> Pay ->
                              Role -> (-> ActorC1, ->
ActorC2).
 deadlineExpires.ActorC1: ClaimExpenses ->
         ApproveHeadDep -> AuthorisedPerson ->
         Pay -> Role -> (-> ActorC1, -> ActorC2).
}

Manifold Process (port in empty_form, port out
completed_form)
{
 begin: //contains the process definition
   (raise startClaim.AssignActivityToRole.
    IF ClaimForm.ClaimAmount < 1000
       raise Approve.AssignActivityToRole
       ELSE raise Approve.AssignActivityToRole.
    IF ApprovalForm.Approoved==YES raise
```

```
Pay.AssignActivityToRole.
  )
 ProcessMonitoring ().
}
Manifold Activity (port in empty_form, competed_form
                   port out empty_form, competed_form)
{
 AssignActivityToRole:
   (raise role.AssignActivityToActor).
 deadlineExpires:
   (IF AreActorsOverloaded()=YES raise
role.DealWithActorsOverload
    ELSE IF ReassignYN==TRUE raise
role.ReassignActivityToActor
         ELSE raise AlertActor.
   )
 ActivityExecuted: // activity finished
  { out in outtray the output form }
}
Manifold Role (port in empty_form, competed_form
               port out empty_form, competed_form)
{
 assignActivityToActor:
  (raise ExamineActorWorkload
   IF NOT Overloaded raise AssignActivityToActor
   ELSE raise DealWithActorOverload
        waits.
  )
 ReassignActivityToActor:
  (raise reAssignActivityToAcotr
   waits.
   )
 ExamineRoleActorsWorkload: ()
DealWithActorsOverload:
  (extend deadline
   allocate more workers
   change activity priorities
  )
 AlertActor: ()
 ShallIReassign: ()
}
```

We end this section by visualizing the framework in Visifold ([5]), Manifold's visual interface. Figure 2 below shows how the `Process` coordinates the allocation of activities to actors through `Roles` and how dynamic reallocation of work occurs when `ActorC1` is not able to perform allocated work on time.

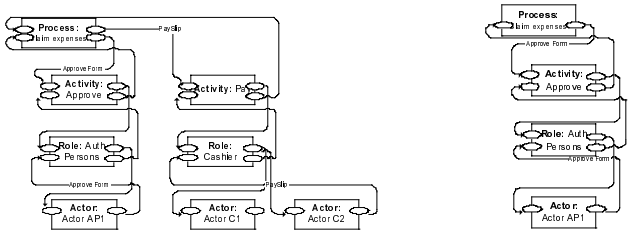

Fig. 2. Hierarchical Allocation and Reallocation of work to actors

5 Discussion; Related and Further Work; Conclusions

In a recent paper, Andrade and Fiadeiro ([2]) argue that Coordination Technologies, as these are understood in the field of Component-Based Software Engineering and Parallel/Distributed Programming, have a contribution to make, in terms of concepts and techniques, to the development of agile Information Systems. The first author of this paper has also argued along the same lines in [15]. Here we elaborate further on the model described in [16] by presenting a two-level hierarchical workflow coordination model. In the process, we have argued for the need to use control-based, event-driven and state-defined coordination programming to model and develop dynamic workflow management systems. We have explained its benefits compared with other approaches that have been used so far, and illustrated its capabilities by means of a specific, if rather simple scenario.

In the short space of a conference paper it would be impossible to describe in detail all the characteristics of our model or compare them in detail with other related approaches. For instance, we have said nothing about examining types of values transmitted via streams (sometimes it may be desirable to know of the data's structure, if not content), etc. This and other issues can be adequately addressed by our model. In particular, our model supports almost all of the functionalities that an adaptive workflow system must exhibit, as those are defined in [10]. More to the point, it supports run-time dynamism, dynamic (re-) configuration, logical decomposition, reusability, and event monitoring.

Over the past few years a number of coordination models and languages have been developed such as Linear Objects (LO), TAO, Gamma and the Chemical Abstract Machine ([17]). However, the first such model, which still remains the most popular one, is Linda ([1]). Although Linda is indeed a successful coordination model, when it is evaluated from the point of view of acting as a framework for modelling human and other activities in information systems, it has some potentially serious deficiencies. The most important deficiency is that it is data-driven i.e. the state of some agent is defined in terms of what kind of data it posts to or retrieves from the Tuple Space. However, there are many cases where we are not interested in the data itself that is being handled; indeed, for security reasons we may not want to allow the examination of data but only coordinate the workflow processes. The issue of security is also relevant in that the medium of communication between processes (the Tuple Space) is an open forum where anyone can post or retrieve tuples. Thus, there is the possibility

of a process either accidentally or deliberately forging, intercepting or stealing information. This has led to the development of models that provide the required security, at the unavoidable cost of increasing the complexity of the model ([14]).

Other related to Linda models are Sonia ([4]) which features the notion of *Agora* ([13]), LAURA ([18]) where the shared space (referred to as *service-space*) is used by agents to post to or retrieve from *forms*, and finally Ariadne ([9]) where the shared workspace is used to hold tree-shaped data and access to them is performed by means of *record templates*.

Our model on the other hand has some clear advantages over the traditional Linda approach and related models:

- Every worker agent is only concerned with getting workload from its input port(s), performing the required work for which it is responsible, and putting the outcome to its out port(s). Such a worker has no need (or way!) to know the environment in which operates and can therefore be substituted with another one without affecting the operation of the rest of co-workers involved.
- Every manager agent is only concerned with making sure that the output produced by some worker agents are sent to some other worker agents that require it. The manager has no need (or way!) of knowing the exact data being transmitted between the worker processes. Thus, security is preserved.
- All entities comprising an activity are treated *homogeneously*. This makes the model very flexible; for instance, new agents can come and go dynamically, some processes may be devices while others may be software programs or humans, etc. The workflow apparatus of our model is not concerned with the nature of the processes being coordinated, only with their input-output inter-dependencies.

We are currently developing a full version of the model as described in this paper, particularly suited to modelling and coordinating activities in distributed information systems, using both a visual ([5]) and textual representation.

References

1. S. Ahuja, N. Carriero, D. Gelernter: Linda and Friends, IEEE Computer 19 (8) (Aug. 1986), 26–34
2. L. F. Andrade, J. L. Fiadeiro: Coordination Technologies for Managing Information System Evolution, CAiSE 2001, Interlaken, Switzerland, LNCS, Vol. 2068. Springer Verlag (4–8 June 2001), 374–387
3. F. Arbab: The IWIM Model for Coordination of Concurrent Activities, First International Conference on Coordination Models, Languages and Applications (Coordination'96), Cesena, Italy, LNCS Vol. 1061. Springer Verlag (15–17 April 1996), 34–56
4. M. Banville: Sonia: an Adaptation of Linda for Coordination of Activities in Organizations, First International Conference on Coordination Models, Languages and Applications (Coordination'96), Cesena, Italy, LNCS, Vol. 1061, Springer Verlag (15–17 April, 1996), 57–74
5. P. Bouvry, F. Arbab: Visifold: A Visual Environment for a Coordination Language, First International Conference on Coordination Models, Languages and Applications (Coordination'96), Cesena, Italy, LNCS Vol. 1061. Springer Verlag (15–17 April, 1996), 403–406

6. N. Carriero, D. Gelernter, S. Hupfer: Collaborative Applications Experience with the Bauhaus Coordination Language, 30th Hawaii International Conference on Systems Sciences (HICSS-30), Mauni, Hawaii, IEEE Press (7–10 Jan., 1997), 310–319
7. M. Cortes: A Coordination Language for Building Collaborative Applications, Computer Supported Cooperative Work, Kluwer Academic Publishers ***9*** (2000), 5–31
8. C. Ellis, K. KeddarA: ML-DEWS: Modeling Language to Support Dynamic Evolution Within Workflow Systems, Computer Supported Cooperative Work, Kluwer Academic Publishers ***9*** (2000), 293–333
9. G. Florijn, T. Besamusca, D. Greefhorst: Ariadne and HOPLa: Flexible Coordination of Collaborative Processes, First International Conference on Coordination Models, Languages and Applications (Coordination'96), Cesena, Italy, LNCS Vol. 1061, Springer Verlag (15–17 April, 1996), 197–214
10. P. T. Kammer, G. A. Bolcer, R. N. Taylor, A. S. Hitomi and M. Bergman: Techniques for Supporting Dynamic and Adaptive Workflow, Computer Supported Cooperative Work, Kluwer Academic Publishers ***9*** (2000), 269–292
11. M. Klein: Challenges and Directions for Coordination Science, Second International Conference on the Design of Cooperative Systems, Juan-les-Pins, France (12–14 June 1996), 705–722
12. T. W. Malone, K. Crowston: The Interdisciplinary Study of Coordination, ACM Computing Surveys **26** (1994), 87–119
13. M. Marchini, M. Melgarejo: Agora: Groupware Metaphors in OO Concurrent Programming, Object-Based Models and Languages for Concurrent Systems, Bologna, Italy, LNCS Vol. 924. Springer Verlag (5 July, 1994)
14. N. H. Minsky, J. Leichter: Law-Governed Linda as a Coordination Model, Object-Based Models and Languages for Concurrent Systems, Bologna, Italy, LNCS Vol. 924. Springer Verlag (5 July, 1994), 125–145
15. G. A. Papadopoulos, F. Arbab: Control-Based Coordination of Human and Other Activities in Cooperative Information Systems, Second International Conference on Coordination Models and Languages, Berlin, Germany, LNCS Vol. 1282. Springer Verlag (1–3 Sept., 1997), 422–425
16. G. A. Papadopoulos, F. Arbab: Modelling Activities in Information Systems Using the Coordination Language MANIFOLD, Thirteenth ACM Symposium on Applied Computing (SAC'98), Atlanta, Georgia, U.S.A., ACM Press (27 Feb.–1 March, 1998), 185–193
17. G. A. Papadopoulos, F. Arbab: Coordination Models and Languages, Advances in Computers, Vol. 46. Marvin V. Zelkowitz (ed.), Academic Press (August 1998), 329–400
18. R. Tolksdorf: Coordinating Services in Open Distributed Systems With LAURA, First International Conference on Coordination Models, Languages and Applications (Coordination'96), Cesena, Italy, LNCS Vol. 1061. Springer Verlag (15–17 April, 1996), 386–402
19. B. C. Warboys, R. M. Greenwood, P. Kawalek: Case for an Explicit Coordination Layer in Modern Business Information Systems Architectures, IEE Proceedings Software, Vol. 146 (3) (June 1999), 160–166
20. S. M. Wheater, S. K. Shrivastava, F. Ranno: OPENflow: A CORBA Based Transactional Workflow System, Advances in Distributed Systems, LNCS Vol. 1752. Springer Verlag (2000), 354–374