Distributed Workflow Enactment: an Agent-based Framework

Giancarlo Fortino, Alfredo Garro, and Wilma Russo

Abstract—**This paper describes the design and the implementation of an Agent-based Workflow Enactment Framework (AWEF) which can be instantiated on the basis of a workflow schema for obtaining a specific workflow enactment engine. A workflow engine therefore is a MAS capable of managing instances of the workflow schema used for the instantiation of AWEF. Each MAS adopts a hierarchical organizational structure composed by an *EnacterAgent*, which is responsible of the activation and monitoring of the workflow, one or more *ManagerAgent*s, which are responsible of the execution and control of the workflow/subworkflows according to a parent/child model, and one or more *TaskAgent*s, which are responsible of the execution of internal tasks and/or of the wrapping of external tasks or services. The hierarchical distribution of the workflow execution control between the *ManagerAgent*s and the distribution of the computation among the *TaskAgent*s allow for more flexible, efficient, and robust enactment services.**

Index Terms—**Multi-Agent Systems, Distributed Workflow Enactment, Workflow Patterns, Agent-based Applications.**

I. Introduction

WORKFLOW Management Systems (WFMS) are systems designed to automate complex activities consisting of many dependent tasks [26]. In the last decades WFMS have been developed to provide support to the modeling, improvement and automation of business management, industrial engineering, and data-intensive scientific processes [25,10]. Since each business area can benefit from workflow management it is possible to distinguish different kind of workflows and related workflow management techniques specifically conceived for meeting the requirements of a specific business area and fully supporting the associated business processes. A main distinction that can be done is between Collaborative and Production-oriented workflows. The former are information centric: human interactions drive the execution of workflows in a loosely structured manner. In this case WFMS are Computer Supported Cooperative Work (CSCW) systems that offer groupware applications and other shared workspace tools for supporting human interactions [4]. Instead, Production workflows are process-driven due to their highly repetitive nature [7]. In this case the processes are highly structured and the adopted WFMS are based on workflow enactment services able to offer an efficient and accurate control about the flow of the processes. Moreover, in a Production workflow, most of the tasks are executed automatically by software programs and applications without interacting with human users. Such a kind of workflows can be modeled as a set of interrelated services (Service Workflow) [29]. In the context of Internet-based workflows [19], such services are distributed and owned by different organizations so that they could become unavailable due to the lack of network service guarantees. To deal with this important issue, a dynamic service allocation of services is often required as well as the negotiation of service level agreements (SLA). Due to these reasons the enactment of Internet-based workflows requires more flexible enactment engines based on more adequate coordination mechanisms.

To effectively fulfill such requirements the Agents paradigm and technology are being used since Agents are widely considered very suitable for the modeling and implementation of complex software systems in open distributed environments [18]. In particular, in the context of workflow management, the use of the Agents paradigm allows for transforming a workflow from a sequence of activities, that are often modeled and consist of (Web) services invocation, in a society of proactive, autonomous and coordinable entities (or multi-agent system) whose coordinated interactions drive the workflow execution [20,17].

This paper proposes an agent-based approach for the distributed enactment of workflows. The workflow enactment is enabled by an Agent-based Workflow Enactment Framework (AWEF) which is instantiated on the basis of the schemas of the workflows to be enacted so obtaining specific workflow engines. A workflow schema can be defined by using the Workflow Patterns identified and proposed by van der Aalst [25] and can be represented using YAWL (Yet Another Workflow Language) [24]. A workflow engine therefore consists of a MAS capable of managing instances of the workflow schema used for the instantiation of the workflow engine. The framework provides the base agents for workflow enactment (*EnacterAgent*, which is responsible of the activation and monitoring of the workflow,

G. Fortino is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: giancarlo.fortino@unical.it).

A. Garro is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: alfredo.garro @unical.it).

W. Russo is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: wilma.russo @unical.it).

ManagerAgent, which is responsible of the execution and control of the workflow, and *TaskAgent*, which is responsible of the execution of internal tasks and/or of the wrapping of external tasks or services), their interaction protocols and a set of control-flow classes which are associated to the behavior of the *ManagerAgent* and implement the Workflow Patterns. The framework is implemented by using JADE [3,16], a FIPA-compliant [13] Java-based agent development environment which basically offers a distributed agent platform and an API for agent programming.

The remainder of the paper is organized as follows. Section 2 introduces some background concepts about workflow enactment, presents our agent-based approach enabled by AWEF and reports some related works. Section 3 and Section 4 describe the design and the JADE-based implementation of AWEF respectively. Finally conclusions are drawn and directions of further work delineated.

II. Distributed Workflow Enactment

The Workflow Reference Model, proposed by the Workflow Management Coalition (WfMC) [28], describes a generic architecture for workflow management consisting of several functional components interfaced with a Workflow Enactment Service (see Figure 1). The Process Definition Tools allow a process designer to define business processes often adopting a diagrammatic representation. The diagrams (or workflow schemas), represented in a Process Description Language (PDL), are received by the Workflow Enactment Service via Interface 1. The Workflow Client Application is usually the application which requests the enactment of a workflow to the Workflow Enactment Service by specifying the workflow schema to be enacted and passing the parameters (Case Activation Record) needed for the activation and execution of a specific workflow instance. During its enactment a workflow can be administered and monitored (Interface 5) and it may interact with other automated business processes (Interface 4), with human participants (Interface 2) and with other applications without human intervention (Interface 3).

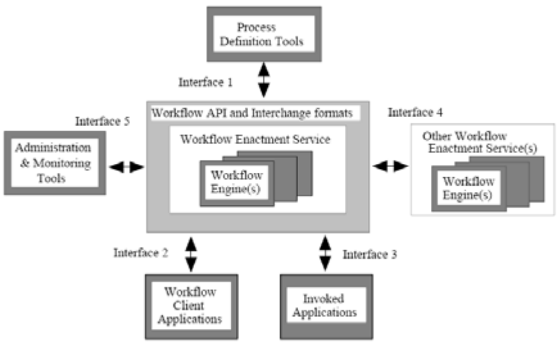
Process Definition Tools
Interface 1
Workflow API and Interchange formats
Interface 5
Interface 4
Administration & Monitoring Tools
Workflow Enactment Service
Workflow Engine(s)
Other Workflow Enactment Service(s)
Workflow Engine(s)
Interface 2
Interface 3
Workflow Client Applications
Invoked Applications

Fig. 1. The reference model for WFMS proposed by the WfMC.

TABLE I
The Workflow Patterns

Pattern Type	**Pattern Name (*Synonyms*)**	**Definition**
Basic Control Flow	Sequence (*Sequential routing, serial routing*)	An activity is enabled after the completion of another activity.
	Parallel Split (*AND-split, parallel routing, fork*)	A point in the workflow where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.
	Synchronization (*AND-join, rendezvous, synchronizer*)	A point in the workflow where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads.
	Exclusive Choice (*XOR-split, conditional routing, switch, decision*)	A point in the workflow where, based on a condition, one of several branches is chosen.
	Simple Merge (*XOR-join, asynchronous join, merge*)	A point in the workflow where two or more alternative branches merge without synchronization.
Advanced Branching and Synchronization	Multi-choice (*Conditional routing, selection, OR-split*)	A point in the workflow where, based on a condition, a number of branches are chosen.
	Synchronizing Merge (*Synchronizing join, OR-join*)	A point in the workflow where multiple paths converge into one single thread.
	Multi-merge	A point in a workflow where two or more branches reconverge without synchronization. If more than one branch gets activated the activity following the merge is started for every activation of every incoming branch.
	Discriminator (*N/M or partial join*)	A point in a workflow that waits for a number of the incoming branches to complete before activating the subsequent activity; then it waits for the remaining branches to complete and "ignores" them. Then it resets itself.
Structural	Arbitrary Cycles (*Loop, iteration, cycle*)	A point in a workflow where one or more activities can be done repeatedly.
	Implicit Termination	A given sub-workflow should be terminated when there is nothing else to be done.
Multiple Instances	Multiple Instances without synchronization (*Multi-threading without synchronization, spawn off facility*)	Multiple instances of an activity can be created with no need to synchronize them.
	Multiple Instances with a priori design time knowledge	An activity is enabled a number of times known at design time. Once all instances are completed some other activity needs to be started.
	Multiple Instances with a priori run-time knowledge	An activity is enabled a number of times known at run time. Once all instances are completed some other activity needs to be started.
	Multiple Instances without a priori run-time knowledge	An activity is enabled a number of times known neither at design time nor at run-time. Once all instances are completed some other activity needs to be started.
State-based	Deferred Choice (*External choice, implicit choice, deferred XOR-split*)	It is similar to the exclusive choice but the choice is not made explicitly and the run-time environment decides what branch to take.
	Interleaved Parallel Routing (*Unordered sequence*)	A set of activities is executed in an arbitrary order decided at run-time; no two activities are active at the same time.
	Milestone (*Test arc, deadline, state condition, withdraw message*)	The enabling of an activity depends on the workflow being in a given state, i.e. the activity is only enabled if a certain milestone has been reached which did not expire yet.
Cancellation	Cancel Activity (*Withdraw activity*)	An enabled activity is disabled, i.e. a thread waiting for the execution of an activity is removed.
	Cancel Case (*Withdraw case*)	A workflow instance is removed completely.

As described above, the Process Definition component provides the process designer with a workflow language able to specify a workflow schema which can be successively instantiated by means of the Enactment Service based on a workflow API and on one or more workflow engines. The workflow definition language is to be expressive and powerful to specify complex workflows from several perspectives: control-flow, data-flow, resource and operational [25].

The control-flow perspective describes activities and their execution ordering through different constructors, which permit to control the flow of execution, and provides an essential insight into the effectiveness of a workflow specification. The data flow perspective rests on control-flow perspective, while the resource and operational perspectives are ancillary.

An expressive and powerful set of control-flow constructs for the specification of workflow schemas (WF Schemas) is the set of the Workflow Patterns proposed by van der Aalst [25]. In Table I the Workflow Patterns, identified by examining the most known contemporary workflow management systems, are enumerated along with their synonyms and a brief definition.

An example WF Schema based on the WF Patterns and drawn by using YAWL [24] is given in Figure 2. After the task A is carried out (sequence pattern), the tasks B, C, and D are executed in parallel (parallel split pattern). When B or C complete (synchronizing merge pattern), the task E is executed an arbitrary number of times (arbitrary cycles pattern). When D completes (sequence pattern), either the task F or the task G (exclusive choice pattern) is executed. When either F or G complete (simple merge pattern), depending on the precedent choice, the task H is executed (sequence pattern). When the iterative execution of E completes and also H terminates (synchronization pattern), the task I is executed and, after its completion (sequence pattern), the workflow terminates.

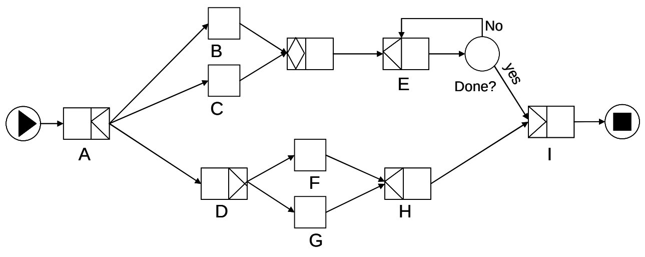

Fig. 2. An example WF Schema based on the WF Patterns.

A generic Workflow Enactment Service (WFES) receives from the user the indication about which workflow is to be enacted (WF Type param) and the Case Activation Record (CAR) for the specific workflow instance; then, on the basis of the WF Schema corresponding to the selected WF Type, the WFES enacts the workflow by means of a specific WF Engine. If the WF Engine is of the distributed type the user indicates also a set of params for specifying some requirements related to the distribution of control, computation and/or data during the workflow enactment (Figure 3).

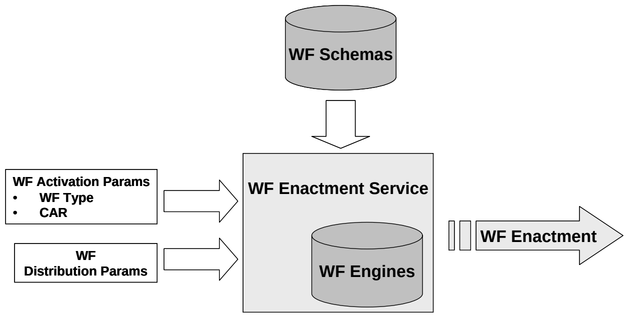

Fig. 3. An A Workflow Enactment Service.

More in details, in order to enact a workflow the WFES selects a suitable WF Engine, from the WF Engines repository, on the basis of the schema of the workflow to be enacted. If the required WF Engine is not available the WFES creates it. The creation is driven by the WF Schema which is used to properly instantiate a Workflow Enactment Framework so building a WF Engine able to enact that specific WF Schema. In building the specific WF Engine the distribution params specified by the user are also considered. Finally, the WF Engine will enact the workflow on the basis of the given CAR and the possible distribution params (Figure 4). The created WF Engine is stored in the WF Engines repository so that it can be reused for enacting future workflows of the same type.

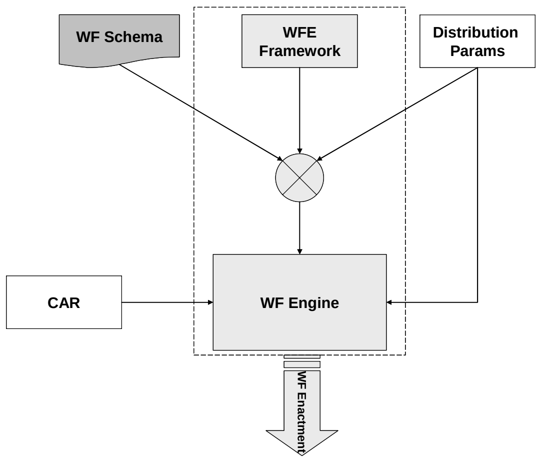

Fig. 4. The construction of a WF Engine.

A. The proposed agent-based approach

In our approach for the distributed workflow enactment a WF Engine is a MAS built by properly instantiating the Agent-based Workflow Enactment Framework (AWEF) on the basis of a WF Schema defined by using the WF Patterns. In particular, a WF Engine consists of tree different agent types:

- *EnacterAgent*, which represents the interface between the MAS constituting the WF Engine and the Workflow

Enactment Service and is responsible for the activation and monitoring of the workflow.

- *ManagerAgent*, which is responsible of the execution and control of the workflow. A single *ManagerAgent* allows for flat workflow management whereas a hierarchical structure of *ManagerAgent*s, formed according to the parent/child model, allows for a hierarchical workflow management. The behavior of a *ManagerAgent* is defined on the basis of the WF Schema it has to enact.
- *TaskAgent*, which is responsible for the execution of internal tasks and/or for the wrapping of external tasks or services. The behavior of a *TaskAgent* is defined on the basis of the activities composing the task it has to carry out.

A WF Engine is, therefore, a MAS with a hierarchical organizational structure in which the control of the workflow execution is hierarchically distributed between the *ManagerAgent*s and the computation is distributed among the *TaskAgent*s.

A WF Schema can be specified by using YAWL [24] which is based on the WF Patterns and offers also an XML based representation of the WF Schema

The design and the implementation of the AWEF, on which the WF Engines are based, are presented in details in sections 3 and 4.

B. *Other related approaches*

In the literature it is possible to find different proposals of distributed workflow enactment mechanisms based on the Agent paradigm and technologies which aim to support more flexible, dynamic and adaptive workflow from the process, resource and activity perspective [8].

Such approaches differ from each other in the supported dimensions of distribution (computation, control and data), in the adopted coordination model (control-driven, data-driven) and in the exploited MAS organizational structure (hierarchical, peer-to-peer).

In [10,23,21] the authors present an agent-based workflow engine centered on a hierarchical organizational structure in which a *ProcessAgent* executes a workflow instance by requesting the execution of the tasks composing the workflow to a set of *ResourceAgent*s. *ResourceAgent*s can be seen as representing web services and can be dynamically discovered and allocated to a *ProcessAgent* by a *ResourceBrokerAgent*. In this control-driven approach the control about the state of the workflow execution is hierarchically distributed between the *ProcessAgent*s and the computation whereas data are distributed among the *ResourceAgent*s which are responsible of the task execution.

In [1] the authors propose a software environment to dynamically generate agent-based workflow engines. A workflow engine is generated by a compiler that translates an XPDL workflow definition to a MAS ready to be executed in the Hermes middleware. The translation process is a two step procedure. In the first step the user-level workflow definition is mapped to an Agent Level Workflow (ALW) specification. In the second step, the compiler concretely generates agents, called Workflow Executors, from the ALW specification by plugging the implementation of the required workflow activities, that are available in a repository, into “empty” agents (skeletons). A workflow engine is, therefore, a MAS having a peer-to-peer organizational structure in which the workflow execution is driven by the interactions among the Workflow Executors.

A similar approach can be found in [22] which presents a methodology for translating a workflow specification into a MAS architecture specifying formalized rules for modeling agents’ behaviors. The MAS is not generated automatically by a compiler like in [1] but by the developer adopting a tool called Agent Developer Studio (ADS).

In [7,8,9] the authors present an agent-based approach for enacting BPEL4WS (Business Process Execution Language for Web Services) [6] workflow specifications. BPEL4WS is an XML-based language that allows for the specification of workflows where the activities are defined by Web service invocations. The proposed distributed enactment mechanisms combine data-centered and control-centered coordination mechanisms. Data are managed via a shared XML repository while the control of the workflow activities is driven by asynchronous messages exchanged between the agents that enact the workflow. The message exchange pattern for the control messages is derived from a Colored Petri Net model of the workflow. The agents’ behaviors are configured and instantiated at run time on the basis of the BPEL4WS specification of the specific workflow to be enacted. The organizational MAS structure is based on a *RequestorAgent* that orchestrates a set of Distributed Workflow Agents according to the workflow specification. The system has been implemented in JADE.

Another agent-based approach for enacting workflows specified in BPEL4WS is proposed in [14]. The novelty of the approach is that the enactment of the workflows is carried out by peer agents that can be associated with web services. The control flow is coded in an interaction protocol that is not defined at the development time like in [1,22] but which is passed at run time between the agents together with the messages so informing each agent what to do next to keep the workflow executing.

Another peer-to-peer agent-based enactment approach is presented in [29]. In this approach the workflow to be enacted is decomposed into a set of interrelated task partitions. Each task partition represents a service and its position, i.e., the interaction and dependency with the other services in the process. Then, each task partition is distributed to an agent which represents a service provider offering a service required by the specific workflow instance. Each agent autonomously manages the enactment of the represented service and the interactions between this service and the others only on the basis of the assigned task partition; agents are not conscious of the whole process in which they are involved. Such adopted coordination model is known as a choreography coordination model.

III. THE DESIGN OF AWEF

The design of AWEF was carried out by exploiting an agent-oriented development process [11] in which the requirements capture phase is supported by the Tropos methodology [5], the analysis and design phases are supported by the Gaia methodology [27] and the detailed design phase is supported by the Agent-UML [2] and the Distilled StateCharts (DSC) [12].

The requirements, captured using the Tropos goal-oriented approach, were reported in a Requirements Statements document. On the basis of the requirements the following key roles were identified:

- *Enacter*, which manages the activation and monitoring of workflows and represents the interface between the WF Engine and the Workflow Enactment Service;
- *Manager*, which manages the execution and control of workflows;
- *Executor*, which executes the internal workflow tasks;
- *Wrapper*, which interacts with the external tasks or services.

Each of these roles was fully described by using a Role Schema according to the Gaia methodology. The protocols associated with each role were identified and documented by an Interactions Model. Then, the identified Roles were aggregated into Agent Types also specifying the agent types hierarchy (Agent Model); the main services required to realize each role were specified (Services Model) and the relationships of communication between the Agent Types documented (Acquaintance Model).

The identified Agent Types are:

- *EnacterAgent*, which derives from the *Enacter* role.
- *ManagerAgent*, which derives from the *Manager* role. A single *ManagerAgent* allows for flat workflow management whereas a hierarchical structure of *ManagerAgent*s, formed according to the parent/child model, allows for a hierarchical workflow management.
- *TaskAgent*, which derives from both the *Executor* and *Wrapper* role.

The detailed design phase allowed for obtaining a detailed specification of the behaviors of the Agent Types which have been defined in the Agent Model. The work products of this phase were the Agent Interactions Model and the Agent Behaviors Model. The former consists of a set of Agent-UML interaction diagrams [2] which thoroughly specify the patterns of interaction between the Agent Types; the Agent Behaviors Model specifies the dynamic behavior of each Agent Type by means of the Distilled StateCharts (DSC) formalism [12].

The main interaction patterns documented by the Agent Interactions Model are:

- *EnacterAgent/ManagerAgent*, which is enabled by the Enacter/Manager Interaction Protocol (EMIP);
- *ManagerAgent/ManagerAgent*, which is enabled by the Manager(parent)/Manager(child) Interaction Protocol (MMIP);
- *ManagerAgent/TaskAgent*, which is enabled by the Manager/Task Interaction Protocol (MTIP).

In the Agent Behaviors Model the basic behaviors of the *EnacterAgent*, *ManagerAgent* and *TaskAgent* are defined. In particular, the defined *ManagerAgent* behavior (or *ManagerBehavior*) is composed of:

- An InitialPseudoActivity, which represents the starting point of the workflow execution in the WF Schema.
- One or more FinalPseudoActivity, which represent points in the WF Schema at which the workflow or a part of it ends. A FinalPseudoActivity uses a parent *ManagerAgent* for notification purposes.
- One or more *WFPattern*, which represent the control-flow activities. A *WFPattern*, which can be any of the available WF Patterns [25] (sequence, and-split, and-join, xor-split, xor-join, or-split, multi-merge, discriminator, loop, multiple instances, deferred choice, milestone, etc) uses one or more *TaskAgent*s and one or more child *ManagerAgent*s for activation purposes and a parent *ManagerAgent* for notification purposes.

In order to model a WF Schema, InitialPseudoActivity, FinalPseudoActivity, and *WFPattern* are linked through source/target control-flow associations.

Figure 5 shows a Statecharts-based representation [15] of the *ManagerBehavior*.

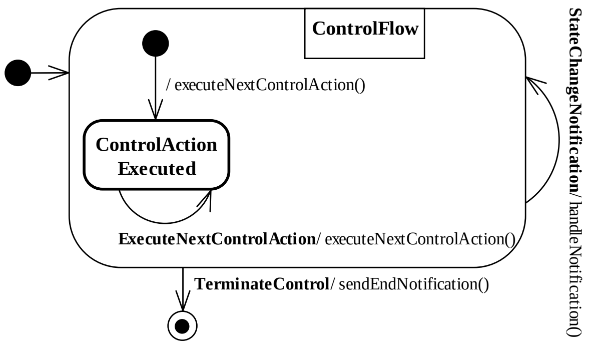
ControlFlow
/ executeNextControlAction()
ControlAction Executed
ExecuteNextControlAction/ executeNextControlAction()
TerminateControl/ sendEndNotification()
StateChangeNotification/ handleNotification()

Fig. 5. The generic behavior of a *ManagerAgent*.

According to the WF Schema to enact, the *ManagerAgent* enters the *ControlFlow* superstate executing the *executeFirstControlAction()* method. In this superstate, every times that an *ExecuteNextControlEvent* is received the *ManagerAgent* executes the next control-flow action by invoking the *executeNextControlAction()* method which fetches the next *WFPattern* and executes it. Upon completion of a *WFPattern* execution, two events are generated: (i) *ExecuteNextControlAction* which allows the *ManagerBehavior* to invoke the *executeNextControlAction()* method; (ii) *StateChangeNotification* which allows notifying the upper-level *ManagerAgent* or the *EnacterAgent* about the control-flow state change of the workflow. If there are no more *WFPatterns* to execute, the *TerminateControl* event is generated which drives the termination of the *ManagerBehavior* and the transmission of the related *EndNotification* to the upper-level *ManagerAgent* or to the *EnacterAgent*. A *WFPattern* execution can involve: (i) the detection of the completion of a task through the reception of a FIPA ACL message which can also carry the data produced by the completed task; (ii) the creation and/or activation of *TaskAgent*s or *ManagerAgent*s.

The interaction diagrams composing the Agent Interactions Model and the behavioral specifications of the Agent Behaviors Model are to be intended as basic schemas that will be coded into the basic classes of AWEF. A WF Engine will be obtained by instantiating such basic classes according to the schema of the workflow to be enacted and using the concrete implementations of the tasks required for the workflow execution.

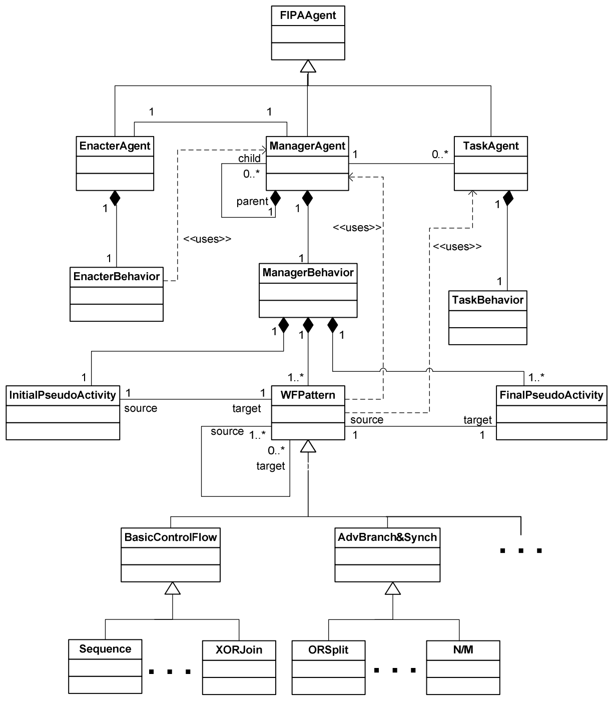

Fig. 6. Class diagram of the AWEF Framework.

In Figure 6 the classes which compose AWEF are reported. In particular, AWEF provides the base agents for workflow enactment (*EnacterAgent*, *ManagerAgent* and *TaskAgent*), their interaction protocols and a set of control-flow classes which are associated to the behavior of the *ManagerAgent* and implement the WF Patterns.

IV. THE JADE-BASED IMPLEMENTATION OF AWEF

The JADE-based classes of AWEF were straightforwardly derived from the class diagram reported in Figure 6. In particular:

- *EnacterAgent*, *ManagerAgent* and *TaskAgent* extend the *Agent* class of JADE [16];
- *EnacterBehavior*, *TaskBehavior* and *WFPattern* extend *Behaviour* class of JADE which represents a generic behavior terminating when the end-of-activity condition is met;
- *ManagerBehavior* extends *FSMBehaviour* class of JADE which models a complex task whose sub-tasks correspond to the activities performed in the states of a finite state machine. In particular, the states of *ManagerBehavior* correspond to the control-flow states of the workflow (or sub-workflow) that the *ManagerAgent* is controlling; each state is associated to a *WFPattern* which is activated when the state becomes active. EMIP, MMIP, and MTIP are appositely defined through sequences of ACL messages instances of the *ACLMessage* class of JADE.

In the following subsection a hierarchical WF Engine based on AWEF and capable of enacting the WF Schema reported in Figure 2 is presented.

A. A hierarchical WF Engine based on AWEF

The hierarchical workflow management is enabled by a set of *ManagerAgent*s each of which embodies a sub-schema of a WF Schema according to a hierarchical model. With reference to the WF Schema of Figure 2, the WF Engine able to enact such a WF Schema is obtained through:

1. The partitioning of the WF Schema into a set of hierarchically arranged workflow schemas: WF Schema 1, WF Schema 1.1, WF Schema 1.2 (see Figure 7);
2. The instantiation of AWEF with respect to the obtained workflow schemas (see Figure 8 for the resulting class diagram).

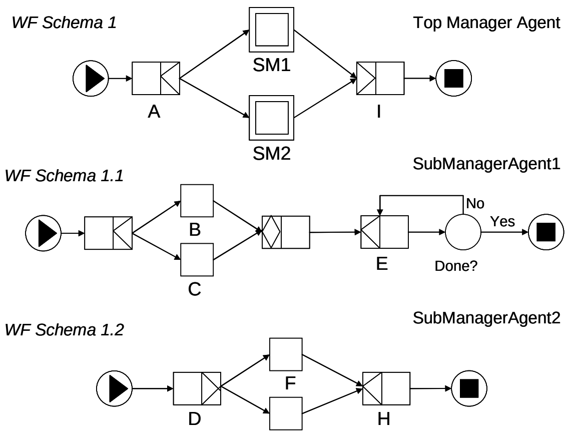

Fig. 7. Partitioned WF Schema.

With reference to Figure 8a the *EnacterAgent* is linked to the top-level *ManagerAgent* which is, in turn, linked to the *TaskAgent*s related to the WF Schema 1, and to the *SubManagerAgents1*and *SubManagerAgent*s*2* which control the WF Schemas 1.1 and 1.2 respectively. Each *SubManagerAgent* is, in turn, linked to the *TaskAgent*s associated to its schema.

With reference to Figures 8b-d each *ManagerBehavior* is obtained by translating its associated WF Schema in a set of classes consisting of one *InitialPseudoActivity*, one or more *FinalPseudoActivity*, and one or more *WFPattern* which are appositely interconnected.

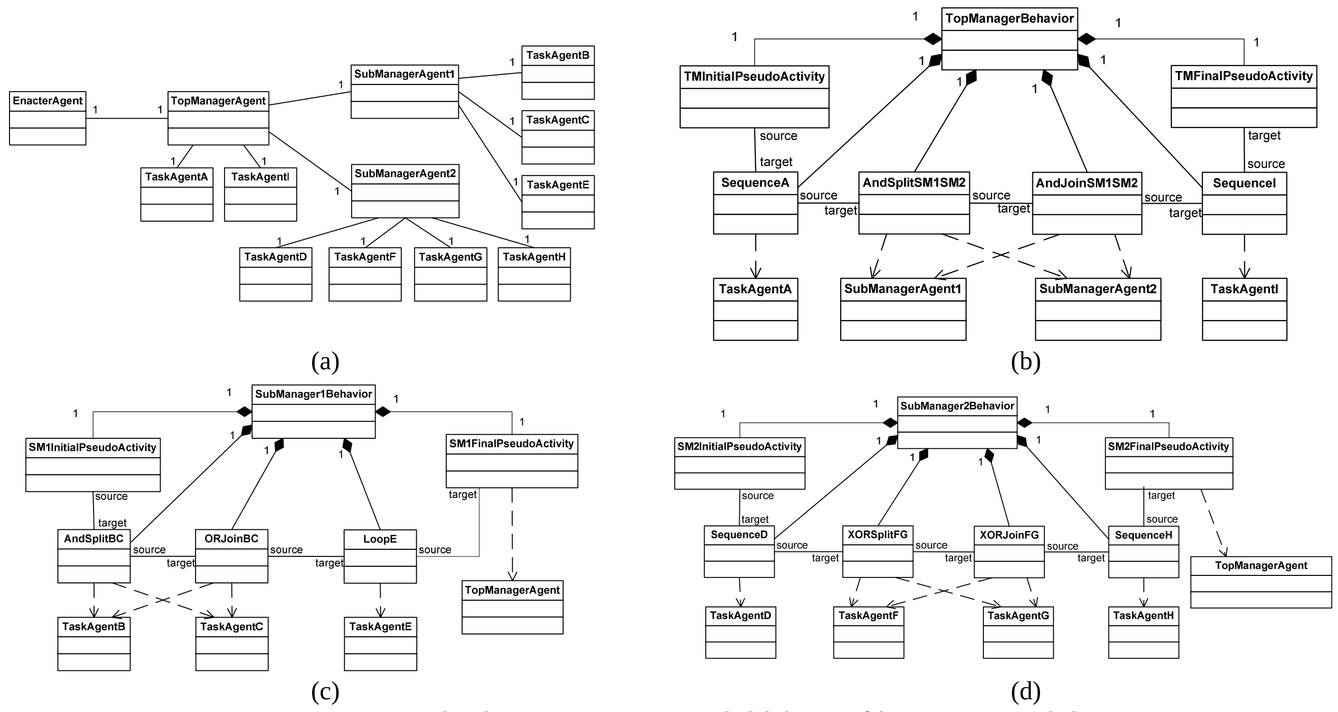

Fig. 8. WF Engine class diagram: (a) MAS structure; (b-d) behaviors of the *ManagerAgent*s (b-d)

V. CONCLUSIONS

This paper has described an Agent-based Workflow Enactment Framework (AWEF) which can be instantiated on the basis of a WF Schema for obtaining a specific WF Engine which mainly consists of a hierarchy of *ManagerAgent*s. Each *ManagerAgent* has in charge the enactment of a sub-schema of the WF Schema used for the instantiation of AWEF and exploits a set of *TaskAgent*s for the execution of the specific workflow tasks associated to its sub-schema. This MAS organization allows for the hierarchical distribution of the workflow execution control between the *ManagerAgent*s and for the distribution of the computation among the *TaskAgent*s. Due to these features AWEF constitutes a basic component for the construction of more flexible, efficient, and robust Workflow Enactment Services.

The JADE-based implementation of AWEF has been applied to the development of a workflow system for the monitoring of distributed agro-industrial productive processes. The developed workflow system is a component of a larger system which was built in the context of the M.ENTE (Management of integrated ENTErprise) project which aims at developing a pervasive system for the control and management of productive, organizational, and business processes of companies working in the agro-alimentary industry of Calabria. The current experimentation of the system provides support to a consortium of agro-industrial greenhouses.

Efforts are currently underway to develop an enactment service which is able to automatically instantiate AWEF on the basis of WF Schemas defined in YAWL.

ACKNOWLEDGMENT

This work was partially supported by the Italian Ministry of Education, University and Research in the framework of the M.ENTE (Management of integrated ENTErprise) research project PON (N°12970-Mis.1.3). The authors also wish to thank Serena Martino and Giuseppe Agapito for their contribution to the JADE-based implementation of the AWEF framework.

REFERENCES

[1] E. Bartocci, F. Corradini, E. Merelli, Enacting proactive workflows engine in e-Science, In proceedings of the 6th International Conference on Computational Science (ICCS 2006), University of Reading, UK, May 28-31, 2006, pp. 1012-1015, volume 3993/2006, Springer-Verlag, Berlin.

[2] B. Bauer, J.P. Muller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Interaction. In Paolo Ciancarini and Michael Wooldridge, editors, *Agent-Oriented Software Engineering*, pages 91-103. Springer-Verlag, Berlin, 2001.

[3] F. Bellifemine, A. Poggi, and G. Rimassa, Developing multi-agent systems with a FIPA-compliant agent framework, *Software Practice and Experience,* 31, pp. 103-128, 2001.

[4] U. M. Borghoff, J. H. Schlichter. Computer-Supported Cooperative Work: Introduction to Distributed Applications. Springer-Verlag. 2000.

[5] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology, *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203-236, 2004.

[6] Business Process Execution Language for Web Services version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[7] P. Buhler, J.M. Vidal, Towards AdaptiveWorkflow Enactment Using Multiagent Systems, *Journal of Information Technology and Management*, vol. 6, pp. 61–87, 2005, Springer-Verlag, Berlin.

[8] P. Buhler, J.M. Vidal, Enacting BPEL4WS specified workflows with multiagent systems, In Proceedings of the Workshop on Web Services and Agent-Based Engineering, 2004.

[9] P. Buhler, J.M. Vidal and H. Verhagen, Adaptive Workflow = web services + agents, In Proceedings of the First International Conference on Web Services, 131-137, 2003.

[10] L. Ehrler, M. Fleurke, M. A. Purvis, and B.T.R. Savarimuthu, Agent-Based Workflow Management Systems(Wfmss): JBees - A Distributed and Adaptive WFMS with Monitoring and Controlling Capabilities, *Journal of Information Systems and e-Business Management*, Volume 4, Issue 1, Jan 2006, Pages 5-23, Springer-Verlag, Berlin.

[11] G. Fortino, A. Garro, and W. Russo, An Integrated Approach for the Development and Validation of Multi Agent Systems, *Computer Systems Science & Engineering*, 20(4), pp.259-271, Jul. 2005.

[12] G. Fortino, W. Russo, and E. Zimeo, A Statecharts-based Software Development Process for Mobile Agents, *Information and Software Technology*, 46(13), pp. 907-921, Oct. 2004.

[13] Foundation of Intelligent and Physical Agents, http://www.fipa.org.

[14] L. Guo, Y. Chen-Burger, and D. Robertson Dave, Enacting the Distributed Business Workflows Using BPEL4WS on the Multi-Agent Platform. In Proceedings of the Third German Conference on Multi-agent System Technologies (MATES2005), LNAI 3550, pp. 35-47, Koblenz Germany, Springer-Verlag.

[15] D. Harel and E. Gery. Executable Object Modelling with Statecharts. *IEEE Computer*, 30(7): 31-42, 1997.

[16] Java-based Agent Development Environment (JADE), documentation and software at the world wide web: http://jade.tilab.com.

[17] N.R. Jennings, P. Faratin, T.J. Norman, P. O'Brien, and B. Odgers, Autonomous Agents for Business Process Management, *Journal of Applied Artificial Intelligence*, 14(2), pp. 145–189, 2000.

[18] M. Luck, P. McBurney, and C. Preist, A Manifesto for Agent technology: Towards Next Generation Computing, *Autonomous Agents and Multi-Agent Systems*, 9(3), pp. 203-252, 2004.

[19] D.C. Marinescu, Internet-based Workflow Management, John Wiley & Sons, Inc., New York, 2002.

[20] P.D. O'Brien and M.E. Wiegand, Agent-based process management: applying intelligent agents to workflow, *Knowledge Engineering Review*, 13(2), pp. 1-14, 1998.

[21] M. A. Purvis, B.T.R Savarimuthu, and M.K Purvis, A Multi-agent Based Workflow System Embedded with Web Services, In proceedings of the second international workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments (COLA 2004), Beijing, China, September 2004, pp 55-62, IEEE/WIC Press.

[22] M. Repetto, M. Paolucci, A. Boccalatte, A Design Tool to Develop Agent-Based Workflow Management Systems, In Proc. of the 4th AI*IA/TABOO Joint Workshop "From Objects to Agents": Intelligent Systems and Pervasive Computing, 10-11 September 2003, Villasimius, CA, Italy, pp. 100-107, Pitagora Editrice Bologna.

[23] B.T.R. Savarimuthu, M.A. Purvis, M.K. Purvis, and S. Cranefield, Agent-Based Integration of Web Services with Workflow Management Systems, *Information Science Discussion Paper Series*, Number 2005/05, ISSN 1172-6024, University of Otago, Dunedin, New Zealand.

[24] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245-275, 2005.

[25] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, Workflow Patterns, *Distributed and Parallel Databases*, 14(3), pp. 5-51, July 2003.

[26] W.M.P. van der Aalst and K. van Hee, Workflow Management: Models, Methods, and Systems, The MIT Press, Cambridge (MA), 2002.

[27] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[28] Workflow Management Coalition, http://www.wfmc.org.

[29] J. Yan, Y. Yang, R. Kowalczyk, and X. T. Nguyen, A service workflow management framework based on peer-to-peer and agent technologies, In Proc. of the International Workshop on Grid and Peer-to-Peer based Workflows co-hosted with the 5th International Conference on Quality Software, Melbourne, Australia, September 19 -21, 2005.