A Repository of Workflow Components for Cooperative *e*-Applications

Massimo Mecella[1], Barbara Pernici[2], Monica Rossi[1], Andrea Testi[1]

[1] *Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"*
Via Salaria 113, 00198 Roma, Italy
{mecella,rossi,testi}@dis.uniroma1.it

[2] *Dipartimento di Elettronica e Informazione, Politecnico di Milano*
Piazza Leonardo da Vinci 32, 20133 Milano, Italy
barbara.pernici@polimi.it

Abstract: Cooperation of processes in different organizations requires the definition of adequate interfaces among processes to model both data exchange and the cooperation logic. In modern *e*-Applications, in order to interconnect *e*-Services of several organizations in a highly dynamic environment, the needs arises for a framework that allows a precise and rapid definition of cooperative processes, and for an infrastructure for change management both at design and at run-time. This work presents a framework for cooperative processes, by defining both a model to represent process interconnection and an architecture for development tools. The implementation of a repository for workflow components is discussed in detail.

Key words: *e*-Service, Workflow Component, Repository

1. INTRODUCTION

The emergence of Internet allows the development of new interaction business paradigms, commonly referred to as *e*-Business. Besides the widespread *e*-Commerce paradigm, there are other contexts where the use of communication networks and of distributed applications can be taken into consideration in order to offer new added value services: for instance, some important initiatives for the definition of what is referred to as *e*-Government are undertaken in many countries (Elmagarmid and McIver 2001, CITU 2000a).

In all these contexts, one of the major problems is to effectively share and integrate services across the Web; such services, commonly referred to as *e*-Services or Web-Services (VLDB-TES 2000), are exported by different organizations as semantically well defined functionalities that allow users and applications to access and perform tasks offered by back-end business applications. The integration of different *e*-Services provided by different organizations allows them to form what is known as virtual enterprise (VE) (RIDE 1999).

As regards the features of an *e*-Service, many definitions are proposed in the literature (VLDB-TES 2000). In this work we consider an *e*-Service as an application component, provided by an organization, which is (Mecella and Pernici 2001):

– *open*, that is independent, as much as possible, of specific platforms and computing paradigms;
– *developed* not only for intra-organization applications, but mainly *for inter-organization applications*, that is to be assembled and reused in a distributed, Internet-based environment;
– *easily composable*; assembling and integrating the *e*-Service in an inter-organization application does not require the development of complex adapters.

An *e*-Application is a distributed application that integrates in a cooperative way the *e*-Services offered by different organizations. *e*-Applications can be both Business-to-Business (B2B) and Business-to-Consumer (B2C) applications, respectively Administration-to-Administration (A2A) and Administration-to-Customer (A2C) applications in case of *e*-Government (Mecella and Batini 2001).

The term Cooperative Information System (CIS) (Brodie 1998, Mylopoulos and Papazoglou 1997) is often used to define a large number of cooperating systems, distributed over large, complex computer and communication networks and working together cooperatively, requesting and sharing information, constraints, and goals. Therefore an *e*-Application is a particular instance of CIS, and *e*-Services are the basic building blocks for new Internet-based CISs.

The integration of different *e*-Services to support centralized, federated and on-the-fly (Benatallah et al. 2000) virtual enterprises and cooperative *e*-Applications requires the development of a complex framework in which the dynamic interchangeability of different *e*-Services is possible in a semi-automatic way. In order to define such a framework, it is necessary to realize some basic elements, that is:

– the definition of a common *conceptual component model* for *e*-Services, mappable on different technological component models. A component can be defined as “a unit of design (at any level), for which a structure is defined, a name identifying the component is associated, and for which design guidelines, in the form of design documentation, are provided in order to support the reuse of the component and to illustrate the context where it can be reused" (Casati et al. 2000b). Specifically a component has well-defined interfaces for the services it provides and for the services it expects from the others, can be composed with other components, perhaps customizing some of its properties, without modifying the component itself. The major obstacle to the establishment of component-based approaches is the need for a common framework, referred to as the component model, that is the definition of "the world in which the component will live in" (Fowler 1997);

– the definition and development of a *repository* in which the conceptual specifications of different components can be stored. The repository should be accessible to all cooperating organizations;
– the definition and development of *interchangeability rules and algorithms* for the discovery of relationships among components and their dynamic and adaptive substitution in complex *e*-Applications;
– the development of a suite of *tools for cooperative architects and designers*, to be used for the management of the different cooperative components and their assembling.

The emerging of different cooperative technologies and platforms, such as OMG Common Object Request Broker Architecture (CORBA, OMG 1998), SUN Enterprise JavaBeans Architecture (EJB, Monson-Haefel 2000) and Microsoft Enterprise .NET Architecture (Trepper 2000), enables the effective development of such a framework, and its deployment on open architectures. Moreover, the widespread use of a standard modeling language for software artefacts, specifically the OMG Unified Modeling Language (UML, OMG 2000a) pushes towards its use for the definition of an open common conceptual component model.

The aim of this work is to present an ongoing work towards the development of this complex framework. Specifically, in this paper we propose a common conceptual component model for *e*-Services based on a particular tailoring of the UML Class Diagrams and UML Statechart diagrams, to define software components for *e*-Services and *e*-Applications; and we describe the development of a repository of such component specifications, accessible through the CORBA technology.

The remainder of this paper is as following. In Section 2, we discuss the concept of cooperative process as the basis for the definition of the common conceptual component model. In Section 3, we introduce our framework, and in Section 4 and Section 5 we detail the component model for *e*-Services and the development of the repository respectively. Section 6 describes a real *e*-Government scenario that provides motivations for our work and the test bed in which we will try our approach. Finally, Section 7 concludes the paper by remarking which other elements of the framework need to be realized.

2. COOPERATIVE PROCESSES

A *cooperative process*, also referred to as *macro process* (Mecella and Batini 2001) or *multi-enterprise process* (MEP, Schuster et al. 2000), is a complex business process involving different organizations. Unlike traditional workflow processes where all the activities concern the same enterprise, in a cooperative process the activities involve different organizations, either because they form together a VE or since they exchange services and information in a coordinated way.

The problem of inter-organization processes has been studied in workflow literature (Grefen et al. 2000) to provide a technological infrastructure to support inter-organization process cooperation. However, recent widespread use of Internet technology produces frequent modifications in collaborations among organizations, specifically frequent changes of *e*-Service providers (Casati et al. 2000a). In such a framework, it is not possible to adopt a strictly coordinated system for inter-organization process cooperation.

On the other hand, in cooperative processes involving several organizations, it is not necessary that all participants in the process have information about all the details of the process in other organizations. As a result, a more flexible way of interconnecting processes is needed, as well as the technological infrastructure that allow dynamic process reconfiguration.

As outlined in the Introduction, the aim of this paper is to introduce a methodological and technological approach for dynamic *e*-Service configuration in an inter-organization process. The proposed approach is based on the concept of *conceptual cooperative workflow specification*.

We define a reference schema for the inter-organization process. This reference schema is an abstract workflow description, which hides the details of process execution in each of the cooperating organizations. On the basis of this reference schema we define *cooperative workflow components* that can interact playing a role in the cooperative process. The workflow components define both the structure and semantics of data exchanged among the cooperating organizations, and the behavioural aspects that specify which interactions are allowed among components and possible sequences of interactions. In this way, single intra-organization activities are aggregated in a higher-level description, which offer the context for them.

We argue that such workflow components are suitable for defining *e*-Services, which are defined by organizations through the analysis of the overall processes they may be involved in. Specifically, the proposed approach can be adopted along two different lines:

– to derive from the agreed schema the internal workflows in each organization providing *e*-Services, according to specific generation rules, such as the ones proposed in van der Aalst and Basten 1999;
– to interface legacy workflow systems and/or legacy information systems, by building workflow components compliant with the static and dynamic specifications defined to interact in the cooperative process.

Different issues need to be tackled in order to adopt the proposed approach; such problems can be divided into two different categories, related (i) to the definition of the framework and (ii) to the methodological and technological issues that each cooperating organization needs to solve when it develops its *e*-Services.

As regards the first category, the issues to be solved are those ones described in the Introduction, that is the definition of the conceptual component model for workflow components and the development of a repository-based infrastructure for their design and management.

As regards the latter category, the issues are both of methodological nature and of technical one; specifically, techniques have to be devised (i) to identify cooperative workflow components and their structure, (ii) to generate single organization workflows that are compliant with the specifications of the workflow components and (iii) to interface existing systems by building interfaces compliant with the inter-organization process specification. Moreover, from the technological point of view, adequate coordination mechanisms for message and data interchange must be developed, to guarantee a correct behaviour of the cooperative process in heterogeneous dynamically changing software environments; techniques and software tools for mapping conceptual workflow specifications to effective software components deployed according to different component models have to be provided.

In the remainder of this paper, we will focus our attention on defining a conceptual component model for workflow components, by addressing the definition of a workflow component specification language based on UML. Then we describe the development of a repository, which is the basic element on which to develop the entire framework and the related methodologies.

As an example, in Figure 1, we show a cooperative process for Goods Procurement (such the one currently set-up in our department). Each time some new goods (such as new workstations, books, etc.) need to be acquired by someone, a purchase order request is issued, and after the Acquisition

Department validates it, a confirmation or a denial is sent to the requester. In the meanwhile, a purchase order is sent to a Seller Organization (with which typically there is some business agreement), in order to have it satisfied. When the Seller Organization is ready with all the goods in the order, a Shipping Agency is involved for the effective shipment of the goods and an invoice is sent back to the Acquisition Department, which in turn issues a payment order. The Seller Organization concludes its process with the reception of the payment order.

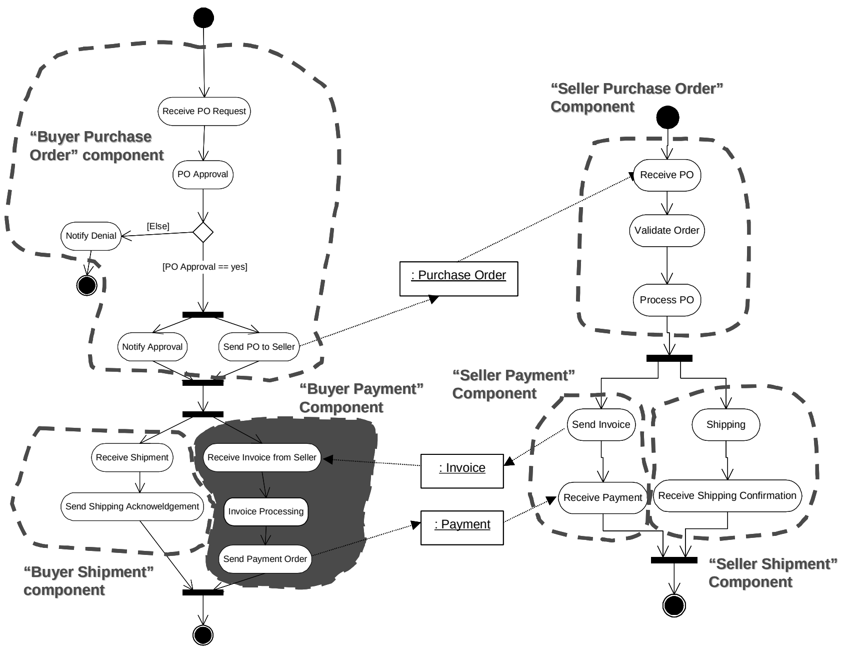

Figure 1. The conceptual specification of the Goods Procurement cooperative process, and the identified workflow components

The two organizations have their own processes (such as the Purchase Order Management and the Invoice&Payment Management in the Buyer Organization), they are separate but a global description of their relationships is needed for a correct cooperation between them. Such a global description has not to consist only on the data (e.g. documents) exchanged, but also on the correct flow of such exchanges, in order to avoid, as an example, deadlock situations.

As shown in Figure 1, our approach describes the conceptual specification of a cooperative process through a UML Activity Diagram. Activity diagrams can be used for modeling cooperative processes at a conceptual level. The purpose of this diagram is to focus on flows driven by internal processing (as opposed to external events, such as in UML Statechart Diagrams).

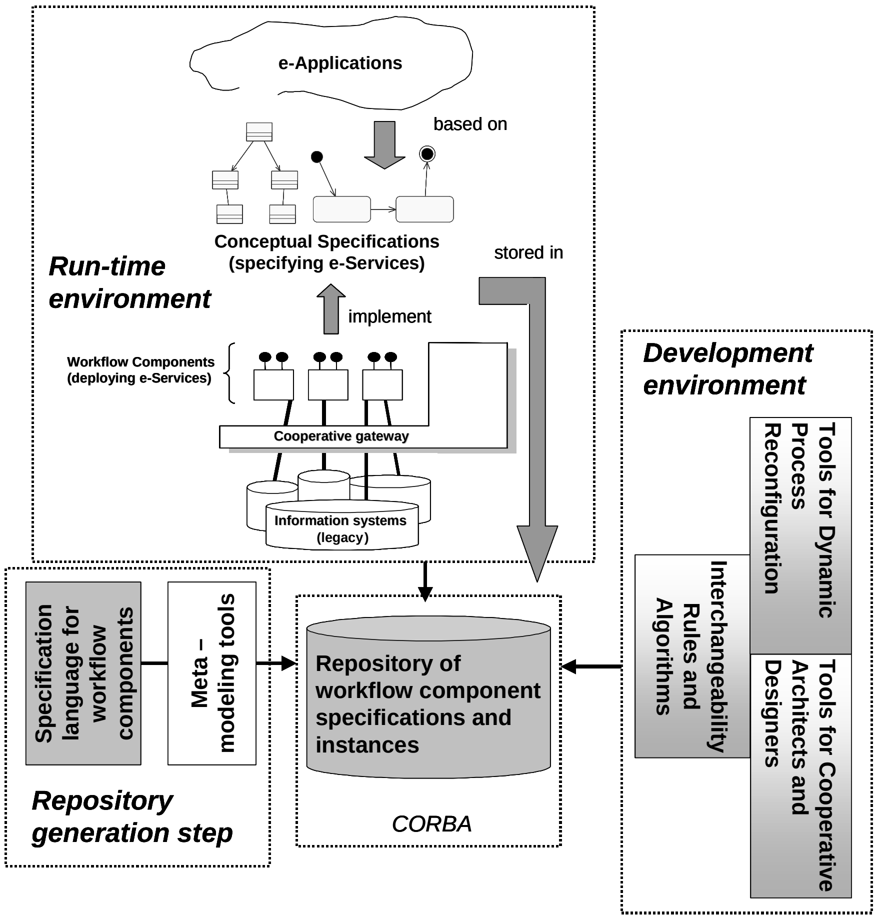
e-Applications
based on
Conceptual Specifications (specifying e-Services)
stored in
Run-time environment
implement
Workflow Components (deploying e-Services)
Cooperative gateway
Information systems (legacy)
Development environment
Tools for Dynamic Process Reconfiguration
Interchangeability Rules and Algorithms
Tools for Cooperative Architects and Designers
Specification language for workflow components
Meta – modeling tools
Repository of workflow component specifications and instances
CORBA
Repository generation step

Figure 2. The proposed framework for *e*-Services as workflow components

The definition of the conceptual cooperative workflow specification is the first step of the joint development process, which aims at defining cooperative workflow components in each participating organization. The next step consists of identifying a set of cooperative workflow components, in order to consider the cooperative process as the exchange of messages among such components. This step identifies on the activity diagram some aggregations of activities, to be assembled and offered as components. Such components are described, in our approach, through UML Class Diagrams and UML Statechart Diagrams, as detailed in Section 4. As an example, in Figure 1 the decomposition in components for the Goods Procurement process is shown. After the cooperative workflow components have been described, each of the organizations needs to provide an implementation of them. This step consists of considering whether some of the activities necessary to carry out the business logic of the components are present in the information system of the organization, and possibly wrapping them according to the conceptual specification of the components, or developing new components according to the specifications.

3. THE FRAMEWORK FOR WORKFLOW COMPONENTS

In this section, we introduce our framework for *e*-Services as workflow components (see Figure 2). The core of the framework is the definition of the conceptual component model for workflow components. Such language is obtained by tailoring UML in a specific way, as illustrated in Section 4.

Once this specification language has been defined, it is possible to develop a repository in which to store both workflow component specifications and instances. The development of such a repository leverages the meta-modeling architecture on which UML was designed and the availability of automatic tools which allow to generate a skeleton of the repository. The technology on which the repository is based is CORBA.

The availability of a repository based on open standards allows an easy integration of different design and run-time management tools to be developed based on this framework. The development environment associated to the repository is based on algorithms for discovering interchangeability relationships among workflow components, and is composed of tools for dynamic process reconfiguration and tools for the specification of a cooperative workflow component, detecting, by accessing the repository, which components can be used to satisfy that specification. If later some components became unavailable, and others have been registered in the meanwhile, the designer can decide to interchange the unavailable ones with the new ones. Such notion can be applied also at run-time, possibly in a semi-automatic (interactive) way.

In most of the approaches, components are considered interchangeable if their interfaces are related by an *is_a* relationship; several papers (Nierstrasz

1995, Harel and Gery 1997, Harel and Kupferman 2000) have been devoted to define inheritance relationships among components based on behavioural aspects. Our interchangeability notion is based on behavioural aspects, and in particular on discovering particular relationships among the statechart diagrams of different component specifications.

Several other tools can be developed or interfaced, based on the repository presented in this paper, in particular graphical CASE tools for the definition and development of new cooperative processes obtained by assembling available components.

As regards the run-time environment, we refer to server organizations as the ones offering their *e*-Services as workflow components, and client organizations as the ones developing new *e*-Applications by assembling different *e*-Services, possibly provided by different server organizations. Each cooperating organization is represented as a domain; each domain exports its cooperative workflow components deploying and making them accessible through cooperative gateways. A cooperative gateway is the computing server platform that hosts the components; different technologies allow the deployment of such elements, and their analysis is out of the scope of the present paper. Specifically, the following layers compose the architecture of each organization:

– Back-End layer (server organizations): where data and applications (possibly) reside. This layer is where the business logic of the workflow components effectively works.
– *e*-Service layer (server organizations): *e*-Services are deployed as workflow components on the server cooperative gateways.
– *e*-Application layer (client organizations): a cooperative process is realized through message exchanges among the workflow components exported by the different organizations. This layer consists of the logic needed to orchestrate such message exchanging.

In the following sections, we detail the conceptual component model and the development of the repository; as regards the run-time cooperative architecture for the different organizations, we refer to previous work of Mecella and Batini 2000 and Mecella and Pernici 2001.

4. WORKFLOW COMPONENT SPECIFICATION

Each *e*-Service is described through the conceptual specification, which consists of diagrams; while the representation of the structural part is easily achieved through UML Class Diagrams, tailored in a specific manner, exporting the behavioural characteristics is more complex, and in the present approach it is obtained through UML Statechart Diagrams. The conceptual specification of a cooperative workflow component consists of:

– A **UML Class Diagram**. The classes represent the specification of both the data exchanged by the component and the services exported by it. These data are both the data the component needs as input and the ones it provides as output. The services it exports are described as events, including both the ones the component reacts to and the events it produces. Therefore the classes represented are of three different categories[1]:
  - *Flat* class: it represents structured data. According to object-oriented terminology, it owns only properties/attributes; the type of each attribute must be either a basic type[2] or a flat class. In the latter case the property is described as an association with the other flat class, and the association end on this side has the name of the property. If the multiplicity is higher than 1, the semantic is that of a multi-set, that is there is not any ordering of the linked flat objects and it is possible to have duplicates. A property/attribute can be optional (meaning that not each instance of the class has to present a value for it).
  - *Active* class: it represents the main functionality provided by the cooperative workflow component. At least one active class needs to be defined in a component; sometimes a component can include the definition of several active classes. An active class has properties/attributes and events.
  - *Event* class: it represents an event an active class reacts to/raises. An event class has attributes/properties; the type of a property can be either a basic type or a flat class. The properties/attributes represent the parameters of the event. An event is linked to the related active classes by the dependency relationship (an active class depends upon an event class) labelled with the stereotype `<<raises>>` or `<<reacts>>`, and to the flat classes specifying its parameters by association relationships.

[1] In the class diagram it is possible to distinguish each category through the three stereotypes: `<<flat>>`, `<<active>>`, `<<event>>`. Every class in the component class diagram needs to be labelled by one of these stereotypes.

[2] We consider as basic types those ones provided by the most common programming languages; specifically our basic types are `Integer`, `Long`, `Single`, `Double`, `Boolean`, `String`, `Date`, `Currency`, `Any`, respectively for numbers (integer, long integer, real in single and double precision), boolean values, strings, date, money and generic (unspecified) values.

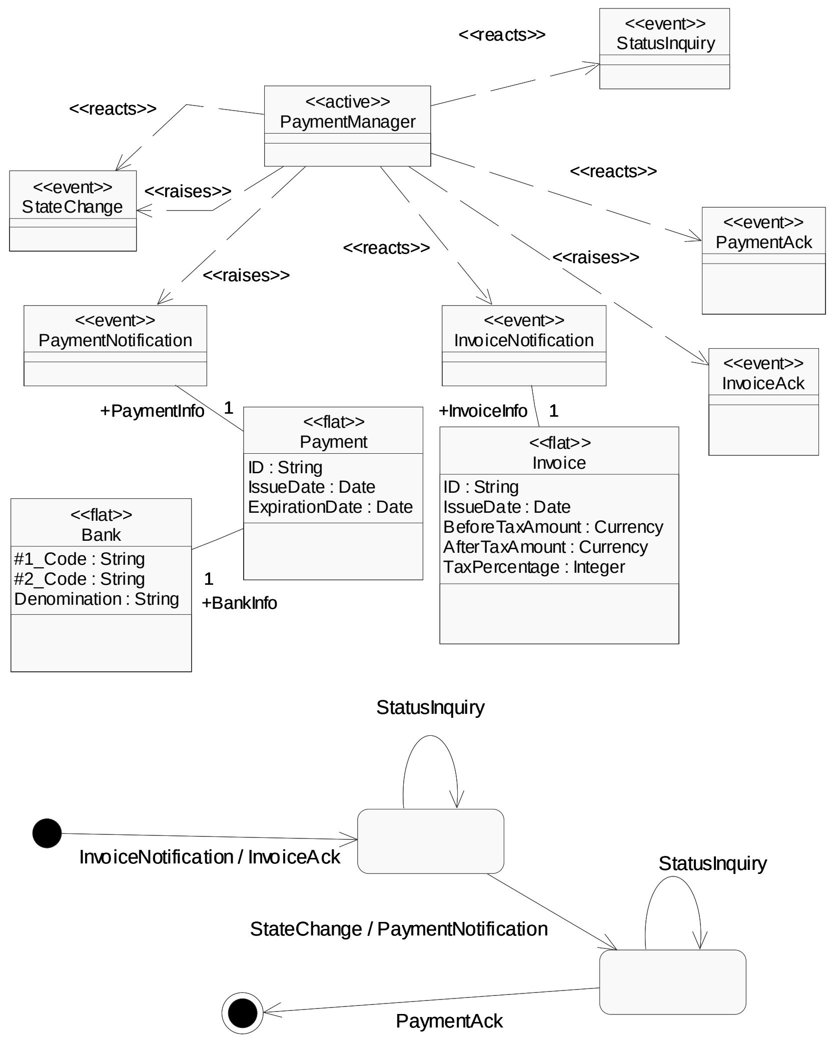

Figure 3. The "Buyer Payment" component specification: the class diagram and the statechart diagram of the only active class `Payment Manager`

– A set of **UML Statechart Diagrams**, one for each active class, which describe how objects (instances of the different classes) evolve during the flow of the cooperative process they are components of. In this work, we use a simplified version of statechart diagrams, without H states, concurrency and nested states.

In Figure 3 the specification of the "Buyer Payment" component is shown.

5. DESIGN AND IMPLEMENTATION OF THE REPOSITORY

With the term repository we refer to an information system that stores, manages and manipulates application definitional information, such as designs, models, component specifications, and so on (Iyengar 1997). Specifically the repository described in this paper allows to store and access to cooperative workflow component specifications expressed according to the language proposed in the previous section.

The repository has been realized using the OMG Meta Object Facility (MOF) conceptual tool (OMG 2000b) and the DSTC dMOF tool (Raymond 2000), which automatizes many steps in the development of repositories.

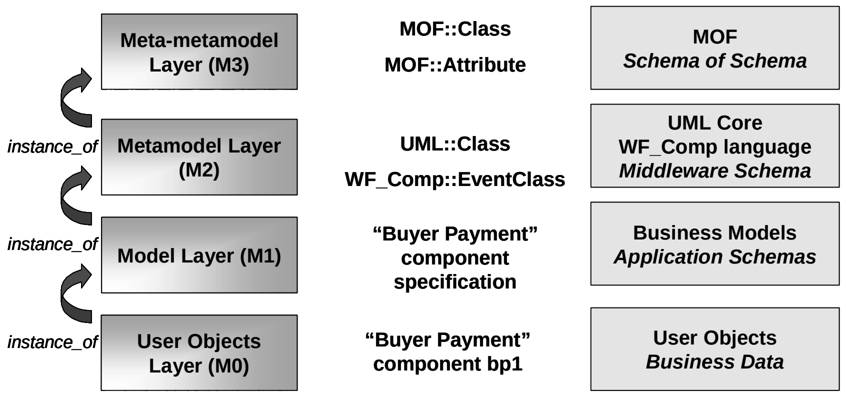

Figure 4. The 4-layer architecture

The 4-layer meta-model architectural pattern (Kobryn 1999) has been adopted as the infrastructure for defining the semantics required by complex models that need to be reliably stored, shared, manipulated and exchanged among tools. The 4 layers, as shown in Figure 4, are the meta-meta-model layer, the meta-model layer, the model layer and the user objects layer. Meta-information is information that describes other information; as an example, a database schema or a class diagrams are meta-information, as they describe how the data items or the software objects are related among them. The universe of entities ("things") in a given domain of interest is referred to as user objects (layer 0), and can be classified into types[3]. A type schema is a collection of types and relationships that jointly describe some "system" of interest according to the particular type language. A particular type schema for a particular system/application is referred to as a model (layer 1). The element in a type schema can be classified into the constructs of the type language (layer 2). Finally, different type languages can be described through a common and general language (layer 3).

The Meta Object Facility (MOF) is a conceptual tool for meta-information management (Raymond 2000); specifically the MOF Model is a semantically well-defined model (layer 3) for describing meta-models (type languages). MOF has been used to describe UML (Kobryn 1999), which is a particular type language.

Our approach consists of using MOF to describe the workflow component specification language introduced in Section 4, which is a particular type language, based on a specific tailoring of UML; a given conceptual specification of a specific workflow component (as the one shown in Figure 3) is actually a type schema.

The DSTC dMOF tool (Raymond 2000) is an implementation of the MOF conceptual tool. In particular it allows the description of a type language, and through an automatic process supported by the tool and shown in Figure 5, to generate a running repository core, deployed as a set of CORBA objects (currently in Java). Such a repository core needs to be extended (by adding specific code in specific plug points) in case that particular operations need to be offered by the repository.

In the dMOF jargon, a MOFlet is the description of a type language, to be expressed in a Meta Object Definition Language (MODL). The `modl2mof` compiler, thus producing an intermediate internal representation, compiles this MODL definition of the type language. From this intermediate output both the IDL interfaces (through the program `mof2idl`) to be successively compiled by a standard CORBA ORB IDL compiler, and CORBA software components (through the program `mof2moflet`) for the type language are produced.

In order to generate such software components, the designer has only to describe the type language in the MODL, as the different dMOF programs automatically do all the generation. This set of CORBA objects constitutes the server side of the repository, that is all the logic for inserting specific workflow component specifications and retrieving them need to be coded as client software invoking the primitives offered by such objects. Moreover

[3] This is the term used in the MOF jargon to refer to classes, entities, software specifications, etc. (Crawley et al. 1997).

the CORBA software objects generated by the tool need to be extended, through specific code to be realized by the designer, if complex access and navigation operations are required.

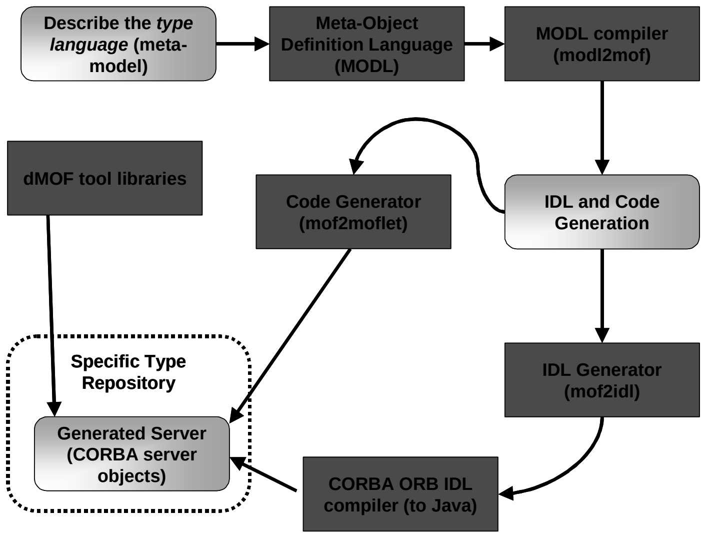
Describe the *type language* (meta-model)
Meta-Object Definition Language (MODL)
MODL compiler (modl2mof)
dMOF tool libraries
Code Generator (mof2moflet)
IDL and Code Generation
Specific Type Repository
Generated Server (CORBA server objects)
IDL Generator (mof2idl)
CORBA ORB IDL compiler (to Java)

Figure 5. The process for generating the repository core through the dMOF tool (from Raymond 2000)

In the case of the type language proposed in this paper, the MOFlet consists of the definition of the specific form of statechart diagram and class diagram described in Section 4. In Figure 6, a portion of the MOFlet is shown; this detail shows the definition of the constructs `State`, `Transition`, `Event` and `Guard`. The Figure 6 is depicted in a graphical notation (derived from the one informally introduced in Raymond 2000) that uses the two basic MOF constructors of class and association[4]. Conversely in Figure 7 the exact MODL specification of the same portion is depicted.

6. AN *E*-GOVERNMENT APPLICATION SCENARIO

The approach presented in this paper is going to be validated in a particular scenario, the Italian *e*-Government initiative (Mecella and Batini 2001).

[4] The other two MOF constructor are data type and package.

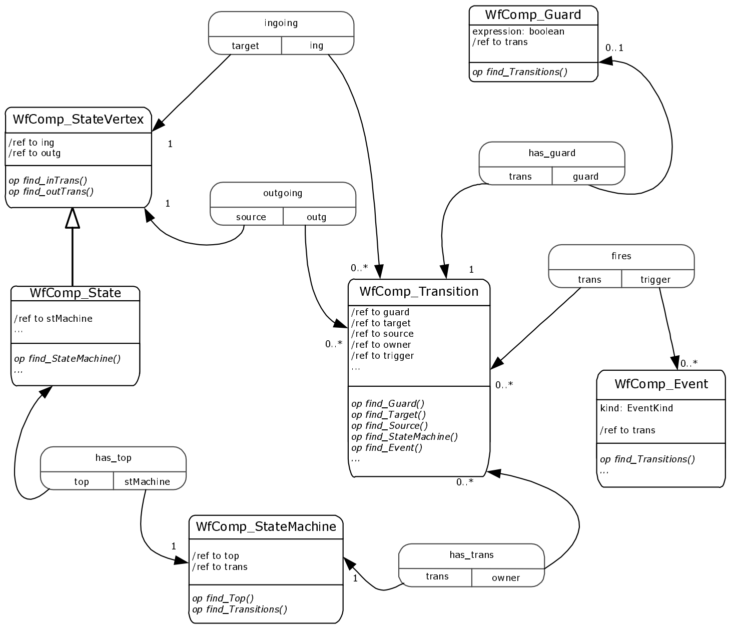

Figure 6. A detail of the MOFlet specification of the type language proposed for workflow component specification: graphical notation

In Italy, the need for a better coordination of efforts and investments in the area of government information systems has pushed, in 1993, the Italian Parliament to create the Authority for IT in the Public Administration (Autorità per l'Informatica nella Pubblica Amministrazione, AIPA) with the aim of promoting technological progress, by defining criteria for planning, implementation, management and maintenance of information systems of the Italian Public Administration (see AIPA's web site for details).

Among the various initiatives undertaken by AIPA since its constitution, the Unitary Network project is the most important and challenging one. The project has the purpose of implementing a "secure Intranet" able to interconnect public administrations. Besides the essential interconnection services (e-mail, file transfer, and so on) which are currently provided to public administrations as basic services, the more ambitious objectives of the Unitary Network will be obtained by promoting cooperation at the application level. By defining a common application architecture, the Cooperative Architecture, it will be possible to consider the set of distributed, yet independent systems of public administrations as a Unitary Information System of Italian Public Administration (as a whole) in which each subject can participate by providing services (*e*-Services) to other subjects (Mecella and Batini 2000, Mecella and Batini 2001).

```
// WfComp.modl
package WfComp
{
//-- Classes --------------------------------------
  abstract class WfComp_Element
  {
    attribute NameType name;
  };

  class WfComp_Transition: WfComp_Element
  {
    reference gua to guard of has_guard;
    reference tar to target of ingoing;
    reference sou to source of outgoing;
    reference own to owner of has_trans;
    reference tri to trigger of fires;

    set [0..1] of WfComp_Guard find_Guard ();
    single WfComp_StateVertex find_Target ();
    single WfComp_StateVertex find_Source ();
    single WfComp_StateMachine find_StateMachine ();
    set [0..1] of WfComp_Event find_Event ();
  };
...
//-- Associations -----------------------------------

  association has_guard
  {
    end single WfComp_Transition trans;
    end set [0..1] of WfComp_Guard guard;
  };
};
```

Figure 7. A detail of the MOFlet specification of the type language proposed for workflow component specification: MODL specification to be compiled by the dMOF tool

The Unitary Network and the related Cooperative Architecture are an example of CIS; specifically all the new inter-administration applications that will be developed on top of the Cooperative Architecture can be considered as *e*-Applications. It is fundamental to establish an overall architecture, yet respecting the autonomy of the single administrations; such autonomy concerns not only the reengineering of the technological systems according to each administration's schedule, but also the reengineering of the various administrative processes providing services to customers. Thus the flexibility of composing *e*-Services of different administrations in order to create new *e*-Applications (supporting complex cooperative processes) must be as high as possible.

Similar initiatives are currently undertaken in the United Kingdom, where the *e*-Government Interoperability Framework (*e*-GIF) sets out the government's technical policies and standards for achieving interoperability and information systems coherence across the UK public sector. To achieve this the government has launched the UK GovTalk initiative (CITU 2000b), that is a joint government and industry forum for generating and agreeing standards, through the definition of XML schemas to be used for information exchange and the development of public portals.

However, the emphasis of these approaches is set on data exchanges, and therefore is focused on document formats (as structural class definitions). In the present approach, the behaviour of workflow components is also represented in the specifications, providing a way to interface process preserving their semantics.

7. CONCLUDING REMARKS AND FUTURE WORK

In the present paper a representation based on UML for describing interchangeable workflow components providing *e*-Services has been presented. A framework to support the definition of workflow components has been discussed.

The framework has a repository of software components as its core element. The realization of the first version of such a repository based on MOF and CORBA technology, using the DSTC dMOF tool, has been presented. The repository will be further developed adding personalization software to add features to manage relationships among component specifications, and between component specifications and their compliant implementations available in the network.

Further research work is ongoing on the basis of the present realization. Theoretical work to define the concept of interchangeability between software components is being developed, with reference to additional services provided by components, and compatibility of parameters between events.

Application interfaces to design tools available on the market to insert components in the repository through graphical user interfaces are being developed. Tools to interface the development tools to run-time environments in different platforms are also being studied, the ultimate goal of the proposed framework being that of providing a repository for workflow components that provides services to organizations working in a heterogeneous distributed environment.

ACKNOWLEDGMENTS

The authors would like to thank Carlo Batini (AIPA), Paola Bertolazzi (IASI-CNR) and Monica Scannapieco (DIS, Università di Roma “La Sapienza”) for important discussions about the work presented in this paper.

The authors would like to thank the Distributed Systems Technology Centre (DSTC), University of Queensland, Australia, and Stephen Crawley for providing the dMOF tool.

Massimo Mecella, Monica Rossi and Andrea Testi would like to thank Microsoft Italia (Area Centro Sud), Roma, Italy, for providing some software needed to carry out the development effort.

REFERENCES

van der Aalst, W.M.P., Basten T.: Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Computing Science Reports 99/06, Eindhoven University of Technology, Eindhoven, 1999.

Autorità per l'Informatica nella Pubblica Amministrazione (AIPA): `http://www.aipa.it/english[4/` (link checked January, 1st 2001).

Benatallah, B., Medjahed, B., Bouguettaya, A., Elmagarmid, A.K., Beard, J.: Composing and Maintaining Web-based Virtual Enterprises. In VLDB-TES 2000.

Brodie, M.L.: The Cooperative Computing Initiative. A Contribution to the Middleware and Software Technologies. GTE Laboratories Technical Publication, 1998. Available on-line: `http://info.gte.com/pubs/PITAC3.pdf` (link checked January, 1st 2001).

Casati, F., Ilnicki, S., Krishnamoorthy, V., Shan, M.C.: Adaptive and Dynamic Service Composition in eFlow. Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAISE 2000), Stockholm, Sweden, 2000.

Casati, F., Castano, S., Fugini, M.G., Mirbel-Sanchez, I., Pernici, B.: Using Patterns to design rules in workflows. IEEE Transactions on Software Engineering, vol. 26, no. 8}, August 2000.

Central IT Unit (CITU) of the Cabinet Office: Information Age Government. Benchmarking Electronic Service Delivery. CITU Report, London, July 2000.

Central IT Unit (CITU) of the Cabinet Office: The GovTalk initiative. `http://www.govtalk.gov.uk/` (link checked January, 1st 2001).

Crawley, S., Davis, S., Indulska, J., McBride, S., Raymond, K.: Meta-meta is Better-better!. Proceedings of the 1st IFIP Working Conference on Distributed Applications and Interoperable Systems (DAIS'97), Cottbus, Germany, 1997.

Elmagarmid, A.K., McIver Jr, W.J. (eds.): Digital Government. IEEE Computer, vol. 34, no. 2, February 2001.

Fowler, M.: Analysis Patterns. Reusable Object Models. Addison Wesley, 1997.

Grefen, P., Aberer, K., Hoffner, Y., Ludwig H.: CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises. International Journal of Computer Systems Science & Engineering, vol. 15, no. 5, 2000.

Harel, D., Gery, E.: Executable Object Modeling with Statecharts. IEEE Computer, July 1997 (also, Proceedings of 18th International Conference on Software Engineering (ICSE '96), Berlin, Germany, 1996).

Harel, D., Kupferman, O.: On the Behavioral Inheritance of State-Based Objects. Proceedings of the 34th International Conference on Component and Object Technology, Santa Barbara, CA, 2000.

Iyengar, S.: Distributed Object Repositories: Concepts and Standards. Proceedings of the 16th International Conference on Conceptual Modeling (ER'97), Los Angeles, CA, 1997.

Kobryn, C.: UML 2001: A Standardization Odyssey. Communications of the ACM, vol. 42, no. 10, October 1999.

Mecella, M., Batini, C.: Cooperation of Heterogeneous Legacy Information Systems: a Methodological Framework. Proceedings of the 4th International Enterprise Distributed Object Computing Conference (EDOC 2000), Makuhari, Japan, 2000.

Mecella, M., Batini, C.: Enabling Italian e-Government Through a Cooperative Architecture. In Elmagarmid and McIver 2001.

Mecella, M., Pernici, B.: Designing Wrapper Components for e-Services in Integrating Heterogeneous Systems. To appear in VLDB Journal, Special Issue on e-Services, 2001.

Monson-Haefel, R.: Enterprise JavaBeans (2nd Edition). O'Reilly 2000.

Mylopoulos, J., Papazoglou, M. (eds.): Cooperative Information Systems. IEEE Expert Intelligent Systems & Their Applications, vol. 12, no. 5, September/October 1997.

Nierstrasz, O.: Regular Types for Active Objects. In Nierstrasz O., Tsichritzis D. (eds.): Object-Oriented Software Composition. Prentice Hall, 1995.

Object Management Group: The Common Object Request Broker Architecture and Specifications. Revision 2.3. Object Management Group, Document formal/98-12-01, Framingham, MA, 1998.

Object Management Group: OMG Unified Modeling Language Specification. Version 1.3. Object Management Group, Document formal/2000-03-01, Framingham, MA, 2000.

Object Management Group: Meta Object Facility (MOF) Specification. Version 1.3. Object Management Group, Document formal/2000-04-03, Framingham, MA, 2000.

Raymond, K.: MOF/XMI Exposed. Keynote and Tutorial Notes of the 4th International Enterprise Distributed Object Computing Conference (EDOC 2000), Makuhari, Japan, 2000.

RIDE 1999: Proceedings of the 9th International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises (RIDE'99), Sydney, Australia, 1999.

Schuster, H., Georgakopoulos, D., Cichocki, A., Baker, D.: Modeling and Composing Service-based and Reference Process-based Multi-enterprise Processes. Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAISE 2000), Stockholm, Sweden, 2000.

Trepper, C.: E-Commerce Strategies. Microsoft Press, 2000.

VLDB-TES 2000: Proceedings of the 1st VLDB Workshop on Technologies for E-Services (VLDB-TES 2000), Cairo, Egypt, 2000.