Building a MultiAgent System from a User Workflow Specification

Ezio Bartocci*, Flavio Corradini*, Emanuela Merelli*
* Dipartimento di Matematica e Informatica,
Università di Camerino,
Via Madonna delle Carceri, 62032 Camerino, Italy
{*name.surname*}@unicam.it

***Abstract*— This paper provides a methodology to build a MultiAgent System (MAS) described in terms of interactive components from a domain-specific User Workflow Specification (UWS). We use a Petri nets-based notation to describe workflow specifications. This, besides using a familiar and well-studied notation, guarantees an high-level of description and independence with more concrete vendor-specific process definition languages. In order to bridge the gap between workflow specifications and MASs, we exploit other intermediate Petri nets-based notations. Transformation rules are given to translate a notation to another. The generated agent-based application implements the original workflow specification. Run-time support is provided by a middleware suitable for the execution of the generated code.**

I. Introduction

Nowadays open distributed systems, characterized by independent components that cooperate to achieve individual and shared goals, are becoming essential in several contexts, from large scientific collaborations to enterprise information systems. The Grid and agent communities are both developing concepts and mechanisms for open distributed systems, but with different perspectives [10]. Grid community has focused on the main prominent cyberinfrastructures [12] for large-scale resource sharing and distributed system integration, providing tools for secure and reliable resource sharing within dynamic and geographically distributed virtual organizations. Grid computing [9] promises users the ability to harness the power of large numbers of heterogeneous, distributed resources such as computing resources, data sources, instruments and application services. Agent community instead is working on the development of methodologies and algorithms for autonomous problem solvers that can act flexibly in uncertain and dynamic environments in order to achieve their goals [13]. As referred in [10], Grid and Agents need each other and are respectively considered the "brawn" and the "brain" of open distributed systems. With the advent of Grid and Agent-based technologies, scientists and engineers are building more and more complex applications to manage and process large data sets, and execute scientific experiments on distributed grid resources. These applications are generally characterized by the execution of a set of distinct, sometimes repetitive, domain-specific activities. Automating such processes requires a model that describes the coordination of the activities to be executed, the roles involved in the organization and the needed resources. For this reason, during the last decade, the workflow technology has became very important. In fact the Workflow Management Coalition (WfMC) defines a workflow *"the automation of a business process, in whole or part, during which documents, information or tasks are passed from one partecipant to another for action, according to a set of procedural rules.*" [21]. In order to provide a proper workflow specification language each standard organization -i.e WfMC, BPMI and OMG- has defined its own process definition language. Our aim is to provide a methodology to translate a User Workflow Specification (UWS) into a MultiAgent System described in terms of Interactive Components [8] (ICs). We have based our approach to describe a MAS with the help of components on that proposed by Ferber in [9] for modelling of MAS in BRIC. Figure 1 shows the two steps of the proposed methodology. In the first step, UWS is translated to a Role-based Workflow Specification (RWS). The user, whose primary expertise is in the application domain, can focus on coordinating domain activities rather than being concerned with the resources involved in the distributed environment. The first translation assigns the resources or roles needed to execute each task. We have chosen Wf-net as high-level specification language suitable to represent the main workflow patterns provided by the most used workflow specification languages -i.e XPDL [22], and BPEL [2]. Wf-net is a well-known extension of classical Petri net [16] notation and it has been introduced in [18]. The second step of the proposed methodology translates RWS into Interactive Components. To describe behavioral aspect of each component we have used BRICs [8] notation; another extension of classical Petri net. We have defined transformation rules to map Wf-net specification patterns into BRICs. This paper is organized as follows.

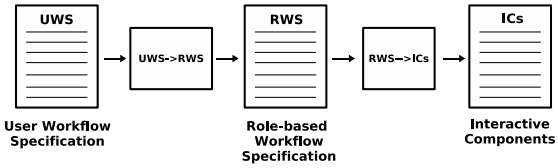

Fig. 1. A two steps methodology

Section 2 describes the background of the work. Section 3 and 4 explain the two steps of our methodology. In Section 5, we present a case study that applies this methodology in Hermes [7] middleware. We conclude in Section 6.

II. BACKGROUND

This section provides some background on Workflow Management System (WMS), Petri nets, High level Petri nets and their application to Workflow Management [18].

A. Workflow Management System

Workflow Management Systems (WMSs) provide an automated framework for managing intra- and interprise business processes. A WMS is defined by WfMC as:*"A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow partecipants and, where required, invoke the use of IT tools and applications.*" [21]. The most part of implemented WMS are based on a client/server architectural style. In these systems, the workflow enactment is entrusted to a central component, that acts as a server and is responsible for the correct execution. These systems lack the flexibility, scalability and fault tolerance required for a distributed cross-organizational workflow; in fact a monolithic architecture does not allow the execution of workflow or parts of it over distributed and heterogeneous systems. To overcome these limitations, agent-based technology promises to alleviate many of these problems [20] and hence enable adaptive workflow. Moreover, using agent mobility, instances of a workflow or parts of it can migrate; i.e., it is possible to transfer the code and the whole execution state, including all data gathered during the execution, between sites participating in workflow's execution. Agent mobility provides two main benefits. First, migrating workflow decreases efficiently traffic network; usually code implementing workflow specification is less heavy to transfer than the amount of data needed during its execution. The second asset concerns the possibility for the workflow to be executed even in mobile and weekly network connected devices. This model requires a suitable middleware to guarantee code mobility support.

B. Petri nets

A Petri net [16] is a directed bipartite graph with two node types called places and transitions. The nodes are connected via directed arcs. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by boxes or bars. According to [15], an *ordinary* Petri net can be defined as a 4-tuple, $PN = (P, T, F, M_0)$ where:

1) $P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places,
2) $T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions,
3) $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),

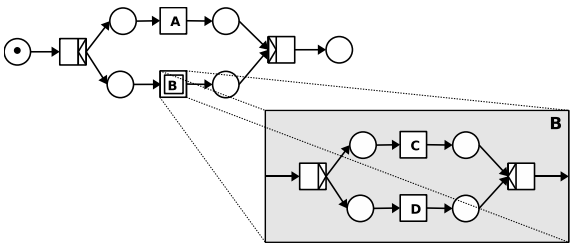

Fig. 2. A subprocess

4) $M_0 : P \to \mathbb{N}$ is the initial marking function,
5) $P \cap T = \oslash$ and $P \cup T \neq \oslash$.

Ordinary means that all arcs have weight 1.

A place p is called an *input place* of a transition t if and only if there exists a directed arc from t to p. Place p is called an *output place* of transition t if and only if there exists a directed arc from p to t. We use $\bullet t$, $t\bullet$ to denote respectively the set of input places and the set of output places a transition t. The notation $\bullet p$ and $p\bullet$ identifies instead the set of transitions sharing p as input place and as output place respectively.

At any time a place contains zero or more tokens, drawn as black dots. A marking function $M \in P \to \mathbb{N}$ is the distribution of tokens over places and represents the state of PN. In this definition we do not consider any *capacity restrictions* for places. The number of tokens may change during the execution of the net.

Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

1) A transition t is said to be *enabled* if and only if each input place p is marked with at least one token.
2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and produces one token in each output place p of t.

C. High level Petri nets

A High-level Petri Net (HLPN) [1] is a PN with three main extensions:

- *Extension with color* - in Coloured Petri Net (CPN) [14] tokens are typed and each token has a value often referred as *color*. Transitions determine the values of the produced tokens on the basis of the values of the consumed tokens. Moreover preconditions can be specified taking into account the color of tokens.
- *Extension with time* - using time extension, tokens receive a *timestamp* value that indicates the time from which the token is available. A token with timestamp 10 is available for the consumption by a transition only from moment 10. A transition is enabled only at the moment when each of the tokens

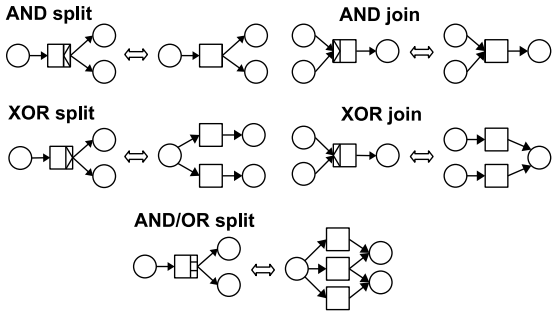

Fig. 3. Special transitions and their translation

to be consumed has a timestamp equal or subsequent to the current time.

- *Extension with hierarchy* - hierarchical extension allows to model complex processes more easily by dividing the main process into ever-smaller subprocesses to overcome the complexity. In this paper, we use the notation proposed by Wil van der Aalst [23], where a subprocess is a transition represented by double-border square as Figure 2 shows.

D. Workflow Nets

Workflow Nets (WF-nets) [23] are a subclass of HLPN where tasks are represented by transitions and conditions by places. A WF-net satisfies two requirements. First of all, it must contain at least two special places: i and o. Place i is a source place with $\bullet i = \oslash$. Place o is a sink place with $o\bullet = \oslash$. Secondly, it must hold that if we add a transition t^* which connect place o with i - i.e. $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$ - then the resulting Petri net is strongly connected -from each node there exists a directed path to every other node-. This requirement avoids dangling tasks and/or conditions. In order to make the WF-net suitable for workflow process modelling a set of notational extensions was applied to the standard Petri net definition. In particular, as referred [23], the author of WF-net added to the classical Petri net transition a set of special transitions (AND split, AND join, XOR split, XOR join, AND/OR split), shown in Figure 3 with their translations, to express branching decisions in a more compact and user friendly way.

In the workflow theory [19], routing primitives are defined as a set possible basic patterns that determine which tasks need to be performed and in which order. Using the prevoius defined special transitions as control flow, a set of four basic routing primitives can be obtained as Figure 4 shows:

1) *Sequential routing* - task A is executed before task B,
2) *Alternative routing* - either task A or task B are executed non deterministically,
3) *Concurrent routing* - task A and task B are executed concurrently,
4) *Iterative routing* - task B is repeated

In order to model dependencies between the workflow process and its operative environment three different constructs named "triggers" were added to the standard Petri net -resource, message and time trigger. In this paper we will consider only the resource trigger. In this particular case, a trigger is associated to a specific resource needed to execute a task. As Figure 5 we can consider a trigger as special place linked with the transition representing a task. When the needed resource is not available this place is empty and the transition is not enabled, while if it contains a token it means that the resource is available and the task related to the linked transition could be executed. In the following sections we will consider interactive components as computational resources able to execute tasks under particular cases. The resource trigger can be assigned to every transition and is represented by a small, self-explaining icon (⇓) near the associated transition symbol as Figure 5 shows.

III. Adding roles to Workflow Specification

A workflow process specification defines which tasks need to be executed and in what order. A set of cases, identified by pre- and postcondition, are handled by executing tasks in a specific order. A task which needs to be executed for a specific case is called *work item* [18]. A workflow specification is the composition of both primitive and complex work items. A primitive work item can be directly executed. A complex work item -called subprocess in [18]- must be specified before it can be used; the specification of a subprocess is a workflow of complex and primitive work items. By using subprocesses the specification of workflows is simplified because they enhance both hierarchical specification and reuse: we can use an already existing subprocess without having care of its specification. Work items are generally executed by a *resource* that can be either a machine -i.e. printer or a fax-, a computational entity -i.e. an agent- or a person. Resources are allowed to deal with specific work items. Grouping resources into classes facilitates the allocation of work items to resources. A resource class based on the capabilities of its members is called *role*. A work item which is being executed by a specific resource is called an *activity*. A workflow designer, whose primary expertise is generally in the application domain, should be free to focus on coordinating domain specific activities rather than being concerned with the complexity of a domain specific activity or resources involved to execute it. Users in fact may ignore the topological organization of the distributed environment and resource classes available. The first step of the proposed methodology translates a user workflow specification to a role-based workflow specification. During this step each work item is assigned to a role able to perform it. This operation could be made manually or automatically. In the first case an expert user can assign role by itself, while in the second case an activity repository store all informations about complex activities and the user knows only there is an automatic mapping from domain specific work items and activities. This resource allocation is applied recursively in all work items of each subprocess. Figure 6 shows an example in

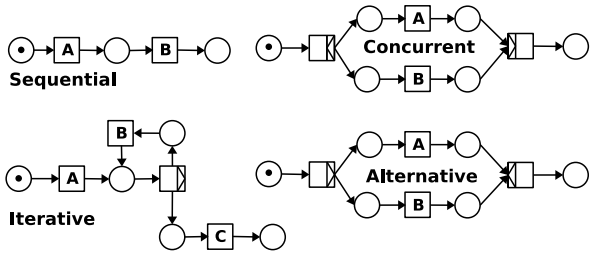

Fig. 4. Routing primitives

bioinformatics. In this case a bioscientist has designed an in-silico experiment -shown on the top of the Figure 6- to globally align some omologous sequences to a given one. This workflow involves five main *work items*:

1) *get_gene_seq* - given a gene id, retrieve the gene DNA sequence,
2) *search_genbank_omologous* - given a DNA sequence, retrieve a set of DNA sequence omologous from NCBI Genbank [3],
3) *search_PDB_omologous* - given a DNA sequence, retrieve a set of DNA sequence omologous from the Protein Data Bank (PDB)[4],
4) *merge_seqs* - merge two or more set of sequences in a set of sequences,
5) *global_alignment* - given a set of DNA sequences calculate the global alignment

In the Role-based Workflow Specification - shown on the bottom of Figure 6- subprocesses *search_genbank_omologous*, *search_PDB_omologous* and *merge_seqs* are substituted with the corresponding set of primitive work items. Each primitive work item is assigned to a specific role. In this case we have three roles A, B and C. Roles are translated into Interactive Components in the next step.

IV. Interactive Components Specification

In the second step the Role-based Specification is translated into Interactive Components. In order to specify the behaviour of each component indipendently from the corresponding generated code, we use BRICs [8], another Petri nets-based notation. In this section we provide transformation rules to translate Wf-net to BRICs notation.

A. BRICs notation

Block Representation of Interactive Components (BRICs) [8] is an high-level language for the design of MultiAgent systems based on a modular approach. A BRIC component -see Figure 7 (a)- is a software structure characterized externally by a certain number of input and output terminals and internally by a set of components. Every component is an instance of a class, which describes its internal structure. A structured component is defined by the assembly of the its subcomponents. The input terminals of the structured components are linked to the input terminals of the the sub-components and is also possible to combine terminals of the composite components with sub-components as showed in Figure 7 (b). The behaviour of elementary components is described in terms of a Petri net based formalism. The default net formalism normally used in BRIC is coloured Petri nets with inhibitor arcs. Figure 7 (c) represents the general form of a transition. A transition is defined by *entry arcs*, *exit arcs* and *pre-condition* of activation. Entry arcs are carriers of a condition, in the form of the description of a token including variables. When the place contains a token corresponding to this description, the arc is validated. There are three categories of entry arcs:

1) *Standard arcs*, denoted $a_1, \cdots, a_n$, trigger the transition only if they are all validated consuming tokens which act as triggers and deleting them from input places.
2) *Inhibitor arcs*, denoted $i_1, \cdots, i_m$, inhibit the triggering of the transition if they are enabled without deleting tokens from the input place.
3) *Non-consumer arcs*, denoted $b_1, \cdots, b_k$, work as standard arcs, but they don't delete the input tokens.

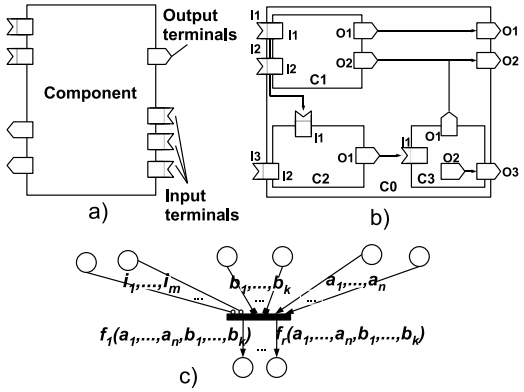

Fig. 7. BRICs notation

An exit arc associate a transition with an output place producing in this position new tokens that depend on the tokens used for triggering the transition. The precondition associated with a transition relates to the external conditions. The components communicate by exchanging information along communication links which connect output terminals to the input terminals. Information is transported through the net in the form of tokens. A token is either an elementary piece of information whose value is a mere presence or absence, or a predicate in the form $p(l_1, \cdots, l_n)$, where each l_i represents a number or a symbol in a finite alphabet. Other importart assumptions concerning this notation are:

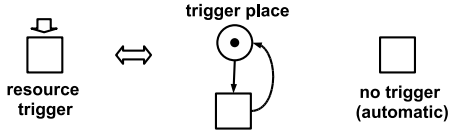

Fig. 5. Resource trigger

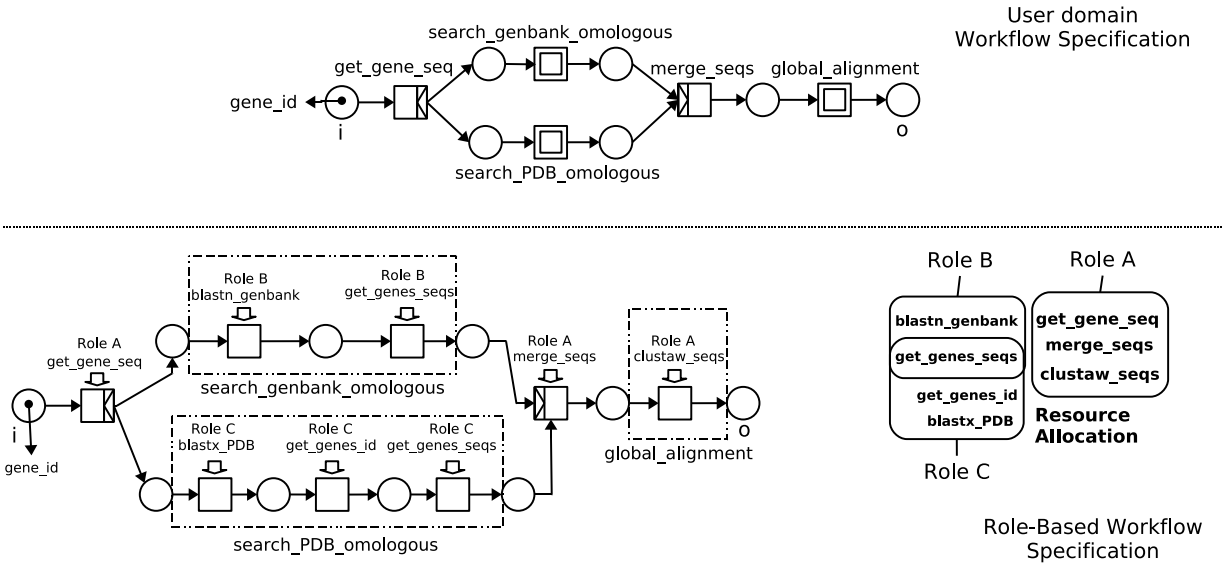

Fig. 6. From User to Role-Based Workflow Specification

1) Input terminals are considered as places, thus names of input terminals are taken to be place identifiers.
2) Any direct link between an input terminal of an incorporating component and an input terminal of incorporated component is assumed to comprise a transition, in accordance with Petri net design rules.

B. Mapping roles with structured components

The translation from a role-based workflow to interactive components specification requires the definition of a structured component skeleton that represents a role-specific implementation. As Figure 8 shows, the basic skeleton has two essential capabilities. First, since it must be able to receive messages from the other external components asynchronously, we specify a subcomponent called *MessagesQueue* that stores messages as coloured tokens following a First In First Out (FIFO) approach. Each message is defined in the form:

`<sender>: <address> << <Act, Pre, Pa>`

where *sender* is the identifier of the component sending the message, *address* is the identifier of the component to which the message is addressed. `Act` and `Pre` are respectively the activity to be chosen and the pre-condition to be set, *Pa* is a possible input parameter for the activity -*null* value means no parameters. In the basic skeleton we specify a second subcomponent, called *Scheduler*, providing, as Figure 9 shows, a set of places and transitions to receive tokens from *MessagesQueue* and to schedule the execution of a set of tasks following the order and cases defined by the role-based workflow specification. Scheduler component has four main places:

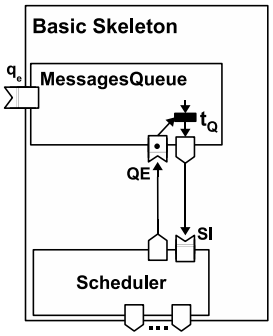

Fig. 8. Basic skeleton component

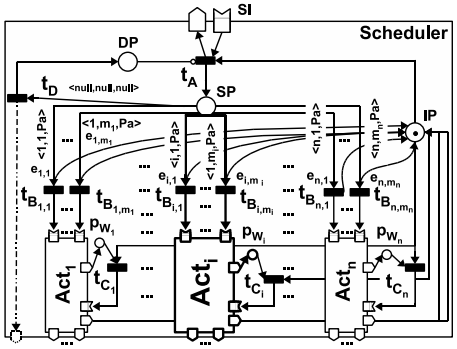

Fig. 9. Scheduler component

1) *Scheduler Input* (SI) - a token in this place means a new message for the scheduler.
2) *Schedule Place* (SP) - after t_A firing produces a coloured token in SP in the form:

   `<Act, Pre, Pa>`

   Each *Scheduler* component contains a set of n Act components and $\forall t_{B_{i,k_i}}$ we define an entry arc e_{i,k_i} with the description:

   `<i, k, Pa>`

   where $1 < i < n$, $1 < k_i < m_i$ and m_i is number

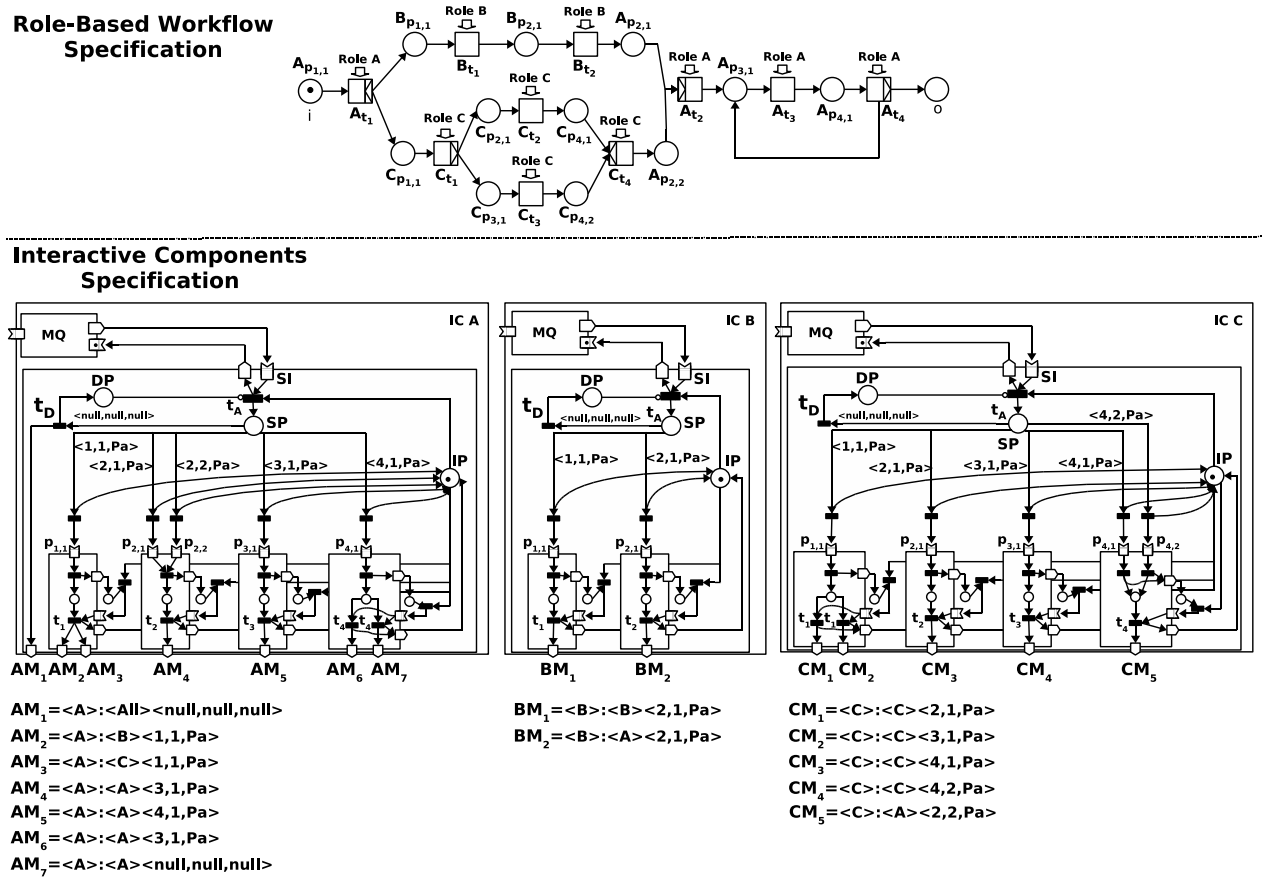

Fig. 11. From Role-Based Workflow to Interactive Components Specification

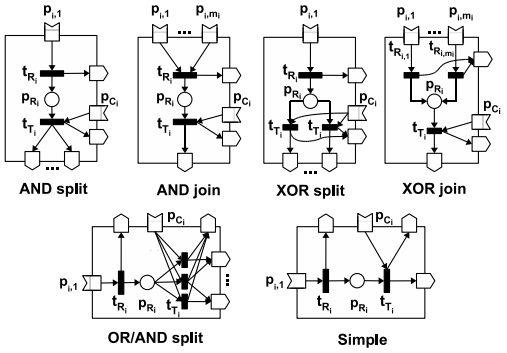

Fig. 10. Mapping activities

of pre-conditions for Act_i. A token in SP matching with a description of an entry arc e_{i,k_i} enables the corresponding transition $t_{B_{i,k_i}}$. The entry arc description for the transition t_D is defined as:

`<null, null, null>`

3) *Idle Place* (IP) - when this place contains a token the *Scheduler* is waiting for a new message.
4) *Dead Place* (SP) - the transition t_D when is enabled produce a token in SP inhibiting the transition t_A. Consequently the *Scheduler* can't receive any token in SP place. This place is called dead, because a token here stops the behaviour of this component. When this happens t_D can produce also a token for the external components to stop their behaviour too:

`<me>: <All> << <null, null, null>`

C. Mapping activities

An Interactive Component (IC) is an executor of a piece of workflow specification. The final behaviour of an IC is obtained by plugging the activities of the corresponding role into the basic skeleton previously defined. Each primitive activity defined in the Role-based specification is associated with an *Act* component in ICs specification. Figure 10 shows how the routing constructs in Figure 4 are mapped into *Act* components. A component Act_i contains an input terminal for each pre-condition of the mapped activities, which are labelled $p_{i,1}, \cdots, p_{i,m_i}$ where m_i is the number of the activity pre-conditions. When the routing transition t_{R_i} fires the token produced in p_{R_i} enable the task transition t_{T_i} -representing a task to be execute by an IC- is enabled iff IP is not empty. The coloured token produced by t_{T_i} is a message -as previously defined- for its and/or other ICs MessageQueue.

D. An example

Figure 11 shows, on the top a Role-based specification using all possibile routing primitives and on the bottom the translation in ICs specification. For each role in the first corresponds to an IC in the second. All pre-conditions and activities are mapped into Act components adding the right routing transitions and are plugged in the Scheduler of the IC basic skeleton. An Act component produce at least a message describing which are the next IC, Act component and input terminal to be reach and an optional parameter for the task transition. The field *address* in the message specifies which are the receiver IC and an entry arc is assumed from this output teminal and the external input terminal of the IC specified.

V. A CASE STUDY

In the previous sections we have defined a Petri nets-based methodology showing how a user workflow-based application specification can be translated into Interactive Components. As a case study we have applied this methodology in Hermes [7], an agent-based middleware, for the design and the execution of activity-based applications in distributed environment. Hermes is structured as a component-based, agent-oriented system with 3-layer -user, system and run-time- software architecture. Due to the lack of space, middleware architecture is not discussed here and we refer to [7] for further details. In this section, we focus instead on its workflow compiler architecture implementing the methodology previously defined. This component, infact, allows to translate a user domain-specific workflow specification into mobile code supported by Hermes middleware.

A. Workflow compilation process

As Figure 12 shows, workflow compilation process in Hermes requires three main components:

1) *WebWFlow* - allows the user to define graphically a workflow of domain-specific activities. A repository provides a set of complex or primitive activities available for selecting. At this level activity implementation details are hidden to user.
2) *XPDLCompiler* - translates the Role-based workflow specification into Interactive Components specification and generates the code to be executed on Hermes middleware. In this case another repository provides the implementation of each activity as a code template.
3) *Hermes middleware* - supports the generated code execution and mobility.

In the follow sections we focus on the first two components details.

B. WebWFlow

WebWFlow is a web-based workflow editor supporting the workflows specification by composing activities in a graphical environment. The graphical notation provided is mapped by WebWFlow into an XML Process Definition

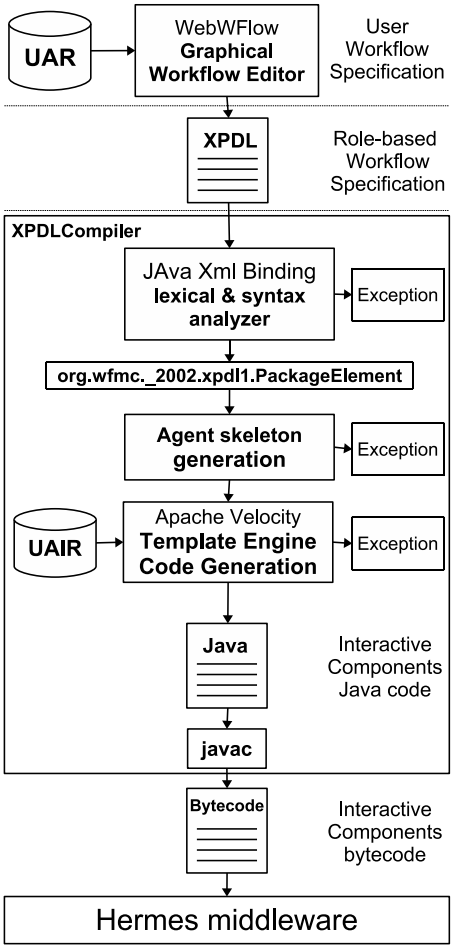

Fig. 12. Workflow compilation process

Language (XPDL) [22] document. WebWFlow allows to import complex activities from the User Activity Repository (UAR). This repository contains the role-based definition of domain-specific activities. The implementation of each activity in UAR is provided instead by the User Implementation Activity Repository (UAIR) and corresponds to a piece of Java code extended with Velocity Template Language(VTL)[11]. The XPDL produced by WebWFlow is a Role-based workflow specification.

C. XPDLCompiler

XPDLCompiler receives an XPDL document and generates the Java bytecode implementing Interactive Components. A lexical and syntax analyzer performs the validation and the parsing of the XPDL document using the Java Architecture XML Binding [17]. After this first phase, the compiler checks if the activities used in the workflow specification have a corresponding implementation in UAIR. Each role is translated in an Agent skeleton, an extension of Hermes UserAgent Java class. As Figure 13 shows, a UserAgent provides the needed communication methods to interact with other UserAgents. Then, for each activity, the corresponding

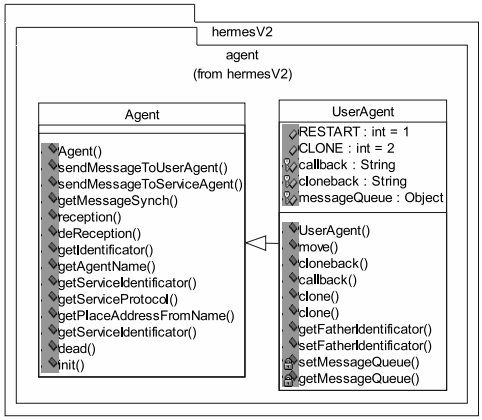

Fig. 13. UserAgent and Agent main methods

implementation code in UAIR is plugged into an Agent skeleton and each internal scheduler is set. The Java code generation is performed using Apache Velocity (http://jakarta.apache.org/velocity/) template engine. Finally, using the Java compiler, the generated bytecode can be loaded into Hermes middleware.

ACKNOWLEDGMENT

This work is supported by the Investment Funds for Basic Research (MIUR-FIRB) project Laboratory of Interdisciplinary Technologies in Bioinformatics (LITBIO). We would also like to thank Luca Tesei for his valuable remarks and suggestions.

VI. CONCLUSION

This paper presents a methodology to build a MultiAgent System described in terms of Interactive Components from a domain-specific User Workflow Specification. The whole approach is described using Petri nets-based notation. This provides many benefits. Petri nets are well-studied formalisms and there are many tools available for verification. The high-level of description provided by Petri Nets guarantees independence with vendor-specific process definition languages. Behaviour of agents can also be described using BRICs, another Petri nets-based notation. In this case it is possible to describe components independently from the their implementing code. Using transformation rules from a notation to another we reduce the gap between workflow specifications and MultiAgent System. Our approach currently supports the building of a MAS based on message passing communication, its extension towards uncoupled communication will be next considered. As future work we also aim to use the approach proposed in [5], [6] to validate the implementation starting from the model.

REFERENCES

[1] W. Aalst. Putting Petri nets to work in industry. *Computers in Industry*, 25(1):45–54, 1994.

[2] T. Andrew, F. Curbera, H. Dholakia, Y. Goland, and et al. Business process execution language (bpel) for web services version 1.1. Technical report, IBM, 2003.

[3] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, and D. Wheeler. Genbank. *Nucleic Acids Res.*, 34:D16–20, 2006.

[4] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, W. H., S. I.N., and B. P.E. Wildfire: distributed, grid-enabled workflow construction and execution. *Nucleic Acids Res.*, 28(1):235–42, 2000.

[5] A. Bertolino, F. Corradini, P. Inverardi, and H. Muccini. Deriving test plans from architectural descriptions. In *ICSE*, pages 220–229, 2000.

[6] A. Bertolino, P. Inverardi, and H. Muccini. An explorative journey from architectural tests definition downto code tests execution. In *ICSE*, pages 211–220. IEEE Computer Society, 2001.

[7] F. Corradini and E. Merelli. Hermes: agent-based middleware for mobile computing. In *Mobile Computing*, volume 3465, pages 234–270. LNCS, 2005.

[8] J. Ferber. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, 1999.

[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[10] I. T. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. In *AAMAS*, pages 8–15. IEEE Computer Society, 2004.

[11] A. Group. Vtl reference guide. http://jakarta.apache.org/velocity/docs/vtl-reference-guide.html.

[12] T. Hey and A. E. Trefethen. Cyberinfrastructure for e-Science. *Science*, 308(5723):817–821, 2005.

[13] N. R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[14] K. Jensen. *Coloured Petri Nets. Basic concept, analysis methods and practical use*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1996.

[15] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.

[16] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Matematik, Bonn, 1962.

[17] Sun. Java architecture for xml binding (jaxb). http://java.sun.com/webservices/jaxb/.

[18] W. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[19] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[20] J. M. Vidal, P. A. Buhler, and C. Stahl. Multiagent systems with workflows. *IEEE Internet Computing*, 8(1):76–82, 2004.

[21] WfMC. Workflow management coalition terminology and glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, 1999.

[22] WfMC. Xml process definition language (xpdl). WfMC standard, W3C, October 2005.

[23] K. v. H. W.M.P. van der Aalst. *Workflow Management - Models, Methods and Systems*. MIT Press, Cambridge, 2002.