Ontology Based Patterns for Collaborative Process Management

Nikos Papageorgiou National Technical University of Athens Iroon Polytechniou 9, 15780 Zografos, Greece +302107721227

npapag@mail.ntua.gr

Yiannis Verginadis National Technical University of Athens Iroon Polytechniou 9, 15780 Zografos, Greece +302107721227 jverg@mail.ntua.gr Dimitris Apostolou University of Piraeus Karaoli & Dimitriou 80 18534 Piraeus, Greece +302104142314 dapost@unipi.gr Gregoris Mentzas
National Technical University
of Athens
Iroon Polytechniou 9,
15780 Zografos, Greece
+302107722415
gmentzas@mail.ntua.gr

ABSTRACT

Collaboration within and between organisations requires knowledge and skills that collaborating partners do not always possess. In an effort to capture best practice collaboration knowledge, we propose patterns as models of repeatable collaboration processes for recurring high-value collaborative tasks. We present a pattern-based approach, associated ontology and tool which act as a platform that can intelligently match collaboration contexts and requirements to collaboration patterns, make intelligent inferences about applying patterns to solve problems at successive levels of abstraction and recommend either workflows or less structured actions as solutions to collaboration contexts.

Categories and Subject Descriptors

D.2.10 [Software]: Design - Representation.

General Terms

Design

Keywords

Ontology, Collaboration Patterns, Process Management

1. INTRODUCTION

Nowadays collaboration is essential for value creation in the modern business environment [1]. Collaboration refers to people or organizations working jointly with others or together especially in an intellectual endeavour that is creative in nature. Collaborative processes may span across organisational and geographical boundaries and may facilitate day-to-day business operations and strategic planning. We have introduced in our previous work the concept of collaboration patterns (CPats) [2;3], as a means for capturing and re-using recurring segments of work or parts of collaboration.

The concept of CPats is inspired by the way experts tackle work on a particular problem: It is unusual to tackle it by inventing a new solution that is completely different from existing ones. Instead, they often recall a similar problem they have already solved, and reuse the essence of its solution to solve the new problem. This kind of 'expert behaviour' is a natural way of

coping with many kinds of problems [4]. The concept of design pattern was first introduced in the field of engineering by Christopher Alexander, a professor of Architecture in University of California, Berkley. His book, where he describes a language for architectural patterns [5], is seen as the prototype for patterns in many other domains, including Software Engineering [6] and Human Computer Interaction (HCI) [7].

In order to process computationally, manage and use CPats we propose their formal representation using the Companion ontology. An ontology is defined as a "formal, explicit specification of a shared conceptualization" [8]. It consists of a set of definitions from a formal vocabulary defining a "schema" and instances, referred to as individuals, of the schema concepts. In a computational context an ontology is a formal, machine readable, shared vocabulary consisting of concepts, relationships, and axiomatic definitions that can be used by standard reasoners to classify and infer new facts.

In this paper we present the Companion ontology in order to formally describe recurring activities that take place in the context of dynamic collaborative environments. Moreover, we aim to take advantage of Companion and develop a dedicated software component that can recommend manage and execute CPats. We base our work on existing research in patterns and on our analysis of requirements of specific case studies. In section 2, we present this pattern ontology along with the CPat model as structured tabular expression of a pattern. In section 3, we argue on how ontology-based patterns can be implemented using an event-based process management framework. In section 4, we discuss related work while section 5 presents our conclusions.

2. ONTOLOGY BASED PATTERNS OF COLLABORATION

2.1 Patterns in Collaborative Work

A CPat is a prescription which addresses a collaborative problem that may occur repeatedly in a business environment [9]. It describes the forms of collaboration and the proven solutions to a collaboration problem and appears as a recurring group of actions that enable efficiency in both the communication and the implementation of a successful solution. A CPat be used as is in the same application domain or it can be abstracted and used as a primitive building block beyond its original domain.

In previous work we defined a CPat structure that comprises all attributes needed for specifying what a specific CPat does, where it is applicable and under which circumstances it may be initiated in a dynamic collaborative environment formulated inside virtual organizations (VOs) [3]. The CPat model encapsulates some of the key findings of the related research in patterns, such as: (a) CPats can serve different objectives or functions and can be of different levels of abstraction; (b) CPats should trigger human and machine processes when certain events occur and condition hold [10]; (c) patterns may include a structure (in the form of a series of steps and the applicable user roles), content (describing the activities to be done), and methods for accessing IT resources to get things done [11]; (d) CPats should contain a diagrammatic description of the proposed solution. In Table 1 we outline the structure of the CPat model in a tabular format.

Table 1. Collaboration pattern model structure

Table 1. Conaboration pattern model structure		
Field	Description	
Name & No:	A name and a number for quick referencing.	
Category:	(Strategic/Business/Simple CPat	
Problem:	A description of the problem(s) the CPat has addressed before or it is expected to address in the future.	
VO lifecycle phase:	One or more VO lifecycle phases where it can be applicable (pre-creation, creation, operation or termination).	
Application Area:	Declares the sector (e.g. Manufacturing) where it is applicable.	
Pre- Conditions:	The list of the states and conditions that must be satisfied before the specific CPat can be considered applicable.	
Triggers:	Events and event patterns that can trigger its execution.	
Triggers of Exceptions:	Events that can raise an exception during implementation of CPat.	
Roles:	Includes the collaboration roles that are to be involved .	
Input Information :	Documents or data that will be used in terms of this CPat.	
Output Information :	Documents or data that will be produced in terms of this CPat.	
Duration:	The acceptable time frame in which the proposed by the CPat solution can be successfully implemented.	
Exception:	A description of an exception to the pattern (e.g. termination of the specific CPat and execution of another one).	
Post- Conditions:	Conditions and states that hold after the successful termination of the CPat.	
Related CPats:	Optional, Alternative, Conflicting CPats	
Solution:	Comprises prescriptions of solutions to the designated problem in the form of action lists, workflows or even instructions for tool usage.	

2.2 Companion: An Ontology for Collaboration Patterns

The aim of Companion, the proposed CPat ontology, is to: (i) provide a formal representation of the CPat model concepts and interrelations, (ii) capture the requirements of prominent collaborative processes, (iii) allow for mapping to collaboration services provided by different suppliers and (iv) provide a flexible structure that can be easily refined, updated, extended and instantiated. As stated by Henninger [11], a major weakness of most pattern representations is the lack of semantics, i.e. typed

relationships between patterns. While other formal media such as UML can be used to model and represent patterns none has the combination of both formal representation and distributed accessibility that ontologies provide. By adopting the arguments of Henninger about the benefits of ontologies for the formal representation of usability design patterns [11], mapped certainly to the requirements of collaboration patterns, we can state that ontologies provide a computational medium that can: (a) intelligently match collaboration contexts and collaborative process requirements to collaboration patterns, (b) make intelligent inferences about applying patterns to solve problems at successive levels of abstraction, thus providing the basis for a pattern language, (c) automatically and dynamically classify patterns into pattern languages that can generate complete design solutions and (d) check the consistency of patterns and pattern language attributes.

Companion is developed in OWL Description Logic (OWL-DL). OWL-DL is a highly expressive yet computable language. We need an expressive ontology language in order to describe CPat terms and preconditions triggers etc. For the development of the CPat ontology we have used the Protégé 3 [12] ontology editor with OntoViz [13] plug-in for visualization and the Pellet reasoner [14] for validation. Figure 1 and Figure 2 depict the pattern concept along with its object properties. Starting from the statement that a Collaboration Pattern is subclass of a Pattern, as described in CPat model, we state that a CPat has Pre-Conditions, Post-Conditions, category (CPatCategory), Application Area, Triggers which are Complex Events and related CPat(s). These relationships are depicted in Figure 1.

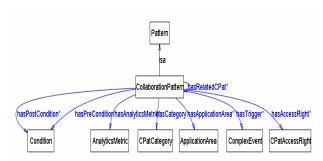


Figure 1 Collaboration Pattern object properties (1 of 2)

Figure 2 shows that a CPat has Input and Output information (CPatInformation), problem(s) (CPatProblem), corresponds to Virtual Organization phase (VOPhase), has Participants (CPatParticipant), has exception(s) (CPatException) and solution(s) (CPatSolution).

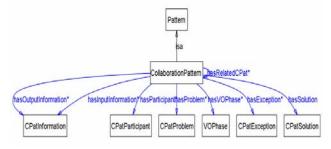


Figure 2 Collaboration Pattern object properties (2 of 2)

Beyond object properties, the class CollaborationPattern has some data properties (CPatDuration, CPatName, CPatNo). All properties are declared as functional, i.e they can have only one, unique value y for each instance x. CPats may be related with other CPats. The object property hasRelatedCPat describes those relationships. Special cases of pattern relationships such as canBeExecutedInSequence, hasAlternativeCPat, hasConflictCPat and canBeExecutedInParallel are modeled with sub-properties.



Figure 3 Collaboration Pattern restrictions

Figure 3 depicts the restrictions of class CollaborationPattern. Every instance of the CPat class:

- is subclass of Pattern and belongs to one of StrategicPattern, BusinessPattern or SimplePattern classes,
- has at least one Access Right, Application Area,
- has exactly one CPatCategory,
- has at least one Participant who is the CPat Initiator,
- has two or more Participants that undertake specific roles in terms of CPat's solution implementation,
- has at least one CPatProblem, CPatSolution, ComplexEvent and VOPhase.

Companion continues with the description of other related CPat concepts. Pre-Conditions and Post-Conditions are of type Condition. A Condition is composed of one or more expressions which evaluate one or more facts. Facts are the elements of a knowledge-base that contain information about the state of collaboration. CPat categories are represented as instances of the CPatCategory class. CPats are also associated with application areas for which we use the North American Industry Classification System (NAICS) [15]. Every CPat and CPat exception can be triggered by a ComplexEvent. A CPat exception involves at least one of the following: the proposed usage of a Collaboration Tool, the execution of an Action List, the execution of another CPat, the execution of a Workflow or a combination of them.

In our approach, every different CPat is represented as a subclass of the CollaborationPattern class and accordingly every collaboration instance of a specific CPat is represented as an instance of the corresponding OWL class. For example in a situation where the collaboration aims to plan, execute and evaluate a kick-off meeting of "Project1" is an instance of the "KickoffMeetingCPat" (owl class) with name "Project1_KickOff_Meeting" (individual). CPat classes can be created with any generic onotology editor or by using a custom editor created for this purpose. CPats, in order to express specific concepts found in CPat attributes, have the possibility to import

the corresponding domain ontologies, e.g. Medical, Geographical, or Virtual Organizations ontologies (Figure 4). In such a way CPats and the tools that process them are not limited to the terminology of Companion but they are free to use any domain ontology in an open and transparent manner.

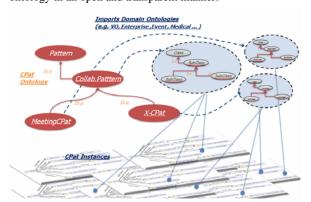


Figure 4 Linking CPats with domain ontologies

2.3 Reasoning with Companion

The most common inferences that are expected to be performed with Companion are the class/subclass and class/individual inferences¹, class inferences may occur when: (i) Axioms are used to assert additional necessary information about a class. (ii) A subclass is inferred due to subclasses being used in existential quantification. (iii) A subclass is inferred due to a subproperty assertion. (iv) There is interaction between an existential quantification (asserting the existence of a class) and a universal quantification (constraining the types of individuals allowed). Instance inferences may occur when: (i) There is an interaction between complete and partial definitions. (ii) There is an interaction between an inverse relationship and domain and range constraints on a property. (iii) The domain restriction gives additional information which allows inference of a more specific type. (iv) There is interaction between an inverse relationship and domain and range constraints on a property. Moreover instance inferences may occur when: (i) There is an interaction between complete and partial definitions. (ii) There is an interaction between an inverse relationship and domain and range constraints on a property. (iii) The domain restriction gives additional information which allows inference of a more specific type. (iv) There is interaction between an inverse relationship and domain and range constraints on a property.

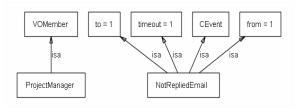


Figure 5: Indicative Companion extension

We give an indicative reasoning example using the extension of Companion shown in Figure 5. With this ontology we define that:

¹ The discussion on class and instance inferencing with ontologies is taken from: http://owl.man.ac.uk/2003/why/latest

(i) The class NotRepliedEMail is subclass of class CEvent (represents complex events), (ii) it has two object properties ("to" and "from") denoting accordingly the sender and the recipient of an email and a datatype property named "timeout" that denotes the duration elapsed before the NotRepliedEmail complex event is generated, and (iii) the ontology contains a class ProjectManager that is subclass of VOMember. Using these concepts we can declare a CPat trigger named ProjectManagementProblem with OWL axioms (Figure 6).

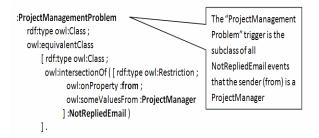


Figure 6: CPat trigger example

This definition enables an OWL reasoner to infer that the NotRepliedEmail events that have sender (pointed by the property "from") a ProjectManager belong to the class ProjectManagementProblem.

```
:NotRepliedEmail 17
:ProjectManager_15
                                   rdf:type:NotRepliedEmail;
 rdf:type:ProjectManager.
                                :from:ProjectManager_15;
                                :timeout "12"^^xsd:int;
:VOMember_14
                                :to:VOMember_14
 rdf:type:VOMember.
                                :NotRepliedEmail_16
:VOMember 13
                                  rdf:type :NotRepliedEmail ;
                                :from:VOMember 13:
 rdf:type:VOMember.
                                :timeout "12"^^xsd:int :
                                :to:VOMember 14.
```

Figure 7: Trigger example instances

By inserting the triples shown in Figure 7 we define two NoteRepliedEmail instances (NotRepliedEmail_16,17) that come from different senders. NotRepliedEmail_17 comes from an individual that belongs to the class ProjectManager (because ProjectManager_15 has rdf:type ProjectManager) but NotRepliedEmail_16 comes from a sender that is not a ProjectManager. As a consequence we expect that only NotRepliedEmail_17 will he inferred as ProjectManagementProblem. A test with Protege and Pellet gives indeed the expected results (Figure 8). Similarly, reasoning with Companion can enable CPat classification, e.g., CPat KickOffMeeting is subclass of CPat_ProjectMeeting because the preconditions, triggers, etc. of the first are subclasses of corresponding properties of the second.



Figure 8: OWL Reasoner results

3. Collaboration Patterns in Event-Based Process Management

3.1 Balancing process flexibility and reusability with CPats

The requirements for the representation formalism of CPats were set during the analysis and modeling phase [3] of our work. This analysis concluded that CPats should be reusable, context-based, flexible, providing solutions adaptable according to the different needs of every specific collaboration instance. To have reusable CPats we need their representation to be abstract enough. On the other hand the need to provide a system able to support automatic, context-based triggering and execution of CPats requires a pattern representation to be transformable to a concrete collaboration instance upon instantiation.

Our approach aims to support the fusion of process-centric with ad-hoc collaboration in an effort to balance reusability and flexibility. Process-centric collaboration is based on pre-defined models that must be fully understood at design-time and enacted at run-time. Models allow for reusability because they can be applied several times as well as for automation because with technologies such as workflow management systems they can be enacted automatically. On the other hand ad hoc, knowledge based collaboration (e.g. situations in which people or businesses must act spontaneously and creatively [16]) requires a means for humans involved in the collaboration to easily define and customize their collaborative actions, at run-time. Balancing flexibility and reusability is necessary in real-life, large scale collaborations in which software services and human actors are involved. In our approach, the catalyst for the envisaged fusion of ad-hoc and process-centric collaboration is Collaboration Patterns (CPats) (Figure 9). CPats facilitate a particular collaboration by providing an encapsulated component that can be reused whenever a collaborative situation/problem occurs [17]. To support ad-hoc as well as process-centric collaboration, the solution needs to involve both user-defined actions and workflows or even collaborative tools for supporting the collaboration.

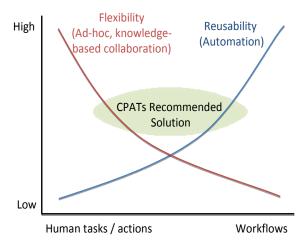


Figure 9: Balancing flexibility and reusability with CPats (adopted from [16])

3.2 CPat Triggering & Execution

CPat triggering is based on conditions and event triggers. CPats are automatically recommended upon the arrival of complex events, if the context is suitable. CPat conditions and triggers are represented again using OWL classes. Classes in OWL can have instances, either set explicitly (asserted) or inferred by their OWL-DL property descriptions. The main idea behind the evaluation of CPat conditions and triggers is that, when the corresponding classes that define the conditions or the triggers have instances, the condition is true or the trigger exists. The evaluation of class instances is based on the execution of an OWL reasoner. The reasoner reads the CPat OWL classes and their property restrictions, evaluates new statements (triples) upon their insertion into the knowledge base and produces inferred triples by translating OWL semantics to rules. By implementing a mechanism that inserts new collaboration knowledge into a CPat knowledge base in the form of OWL statements and combining it with OWL CPats we enable the automatic execution of CPat recommendations and solutions. The following pseudo-code (Table 2) illustrates a CPat triggering logic.

Table 2: CPA Logic

1 ·	WHEN	<trigger></trigger>
1.	IF.	<precondition></precondition>
	THEN	<recommend cpat=""></recommend>
2:	IF	<cpat accepted="" recommendation=""></cpat>
	THEN	<begin (by="" configuration="" cpat="" cpat<="" th="" the=""></begin>
	initiator)>	
3:	IF	<cpat accepted="" recommendation=""></cpat>
	AND	<cpat completed="" configuration=""></cpat>
	THEN	<execute cpat="" solution=""></execute>
4:	IF	<cpat executed="" solution=""></cpat>
	AND	<postcondition true=""></postcondition>
	THEN	<terminate cpat=""></terminate>
5:	WHEN	<exception trigger=""></exception>
	IF	<cpat executed="" solution=""></cpat>
	THEN	<terminate cpat=""></terminate>
	AND	<recommend alternative="" cpat=""></recommend>

Everything starts upon the arrival of a complex event. Then the event is written to the knowledge base in order to be processed by the reasoner and all CPat triggers are examined one by one. For each CPat whose trigger (i.e. the class that is related to the CPat with the hasTrigger property) has some instance, the corresponding CPat PreCondition class is examined. If the precondion is true (has at least one instance) then a CPat recommendation is generated (Table 2 – step 1). The CPat Recommendation is presented by a dedicated tool (CPA – see section 3.3) to all candidate CPat initiators (figure 10).

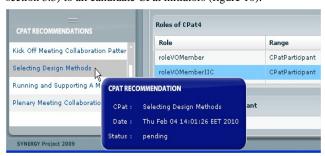


Figure 10: CPat Recommendation

The system automatically discovers CPat initiators by retrieving the instances of the class that is related to the specific CPat with the property hasInitiator (Table 2 – step 2).

CPat initiators have the option to accept or reject a CPat. If a CPat initiator decides to accept a recommended CPat, the CPat Assistant presents a GUI that enables the user to configure the CPat in terms of candidate Participants, Input Information, and Solution (action list or workflow). All invitation and recommendation concepts are expressed with Companion (Figure 11)

Similarly to the CPat initiators, the system proposes CPat participants and input information by retrieving from the knowledge base the instances of the classes that are related to the CPat class with corresponding instances. CPats may have one or more roles. Participants are related to the CPat with properties named role<?role_name> according to their desired role. The property hasInputInformation points to the class that describes a CPat candidate input information. Input information, documents or other data in the knowledge base, are referenced by using a unique URI (e.g. the url of an html document). This URI is an OWL/RDF individual, too. This means that it can be associated with other individuals or literals with OWL properties according to its content. CPat participants may be assigned to a CPat with a specific role either upon accepting an invitation or upon direct assignment by the CPat initiator.

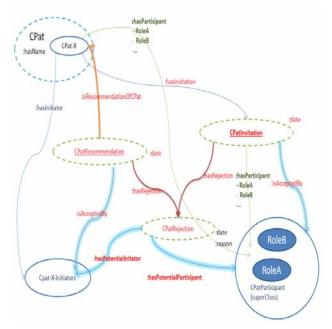


Figure 11: CPat recommendations and invitations

In step 3 (Table 2 – step 3) we define that, when the CPat initiator that accepted the recommendation chooses to initiate the execution of the CPat solution, the system validates the CPat instance (in terms of participants, input information and solution). As a CPat instance is being created all information is stored in the CPat knowledge base in the form of statements using the corresponding properties (hasInputInformation, role<?role_name>, etc) (Figure 12).

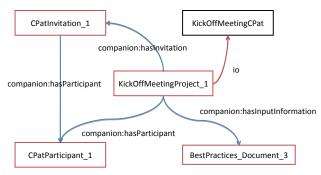


Figure 12: CPat Instance

In step 4, during the execution of the solution and upon the completion of each step the post condition class of the CPat is examined (Table 2 – step 4). If it has instances, the CPat participants are informed that the CPat goals have been reached and the CPat is terminated.

Finally (Table 2 – step 5), for each CPat instance that is currently active (because it has not been terminated for some reason) and whenever a new event is written to the knowledge base, the corresponding CPat Exception Trigger class is being examined for instances. If it is true (because there are some individuals belonging to the relevant class) then the system recommends the termination of the CPat and, in case that an exception CPat has been defined, proposes to start a new CPat in order to handle the exception.

3.3 Collaboration Patterns Assistant (CPA)

The Collaboration Pattern Assistant (CPA) aims to help users execute collaborative processes based on CPats by taking advantage of Companion. CPA tries to enable proactive collaboration support with respect to changing circumstances, by evaluating complex events and facts regarding the collaboration state and deriving recommendations. The users of the CPA, from the point of view of the system, are distinguished in two general roles: (i) CPat initiators and (ii) CPat participants. CPat initiators are responsible for the initiation and the termination of a new collaboration according to the selected CPat. CPat participants use the CPA in order to get information about the current activities inside their group, to participate in collaborations performed within a planned CPat and to get informed about their assigned tasks. CPat initiators may also be CPat participants. The candidate CPat initiators are determined by the system according to the specification of each CPat. CPats are stored in OWL in a dedicated CPat knowledge base. The initiator of a CPat decides the form of collaboration by choosing a loose schema (i.e. adopting an action list) or triggering the execution of a strict workflow.

The CPA prototype is implemented using the Adobe Flex/AIR framework [18] for the client and the open source Flash server Red5[19] (Figure 13). This combination gives us the ability to build on an open source platform which provides facilities very useful in the development of collaborative applications. The communication between the server and the client is done using the RTMP/AMF3 protocols. The use of these protocols allows real-time bidirectional communication between the clients and the server in order to transfer data, commands, events or even streaming audio/video.

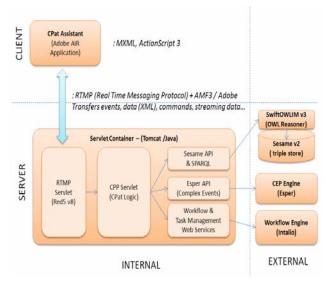


Figure 13: Technical Implementation

The Red5 server is written in Java and runs inside the Tomcat servlet container. The fact that it is written in Java allows easy integration with the existing huge collection of open-source tools and services. All the business logic of CPats is written in Java in the form of a servlet that plugs in the Red5 RTMP servlet. In order to store knowledge we use the Sesame triple-store with the Swift OWLIM plugin for OWL. This combination is very efficient and allows both automatic OWL reasoning (or else ontology materialization) and querying through SPARQL. The incoming events are going to be processed by the Esper [20] complex event processing (CEP) engine. Workflow execution is managed, in the case of the CPA prototype, by Intalio [21]. Intalio is an open source workflow management system which executes workflows written in BPEL, provides human task management and BPEL4People services and a BPMN Editor.

4. RELATED WORK

Our work focuses on supporting ontology and pattern-based collaborations within service-oriented architectures comprising software-enabled, user-enriched services. In such an environment, collaborations typically involve both humans providing their skills and experiences as services, as well as software services thus creating highly dynamic and complex interactions.

Supporting human processes in SOA has been leveraged with technologies such as BPEL4People [22] that target the support of human interactions as part of business processes (i.e., workflows) by designing and executing a set of human tasks, see e.g., WS-HumanTask [23].

Research in patterns has focused on various areas related to collaboration. We identified the most relevant (more than 25 pattern approaches) research and commercial efforts (e.g. Thinklets[24], Usability Patterns[11], Workflow Patterns[25], Service Interaction Patterns[26], e-Business Patterns[27]) to collaboration that take under consideration patterns. We have detected two major high level directions of work. The first direction resembles the detection/mining of patterns in order to observe differentiations from established best practices in

collaboration and propose some manual or automatic corrective actions[26], while the second direction involves all these efforts that focus on describing patterns that are to be executed in order to assist a given collaboration[9]. Due to the rising complexity of the collaborative working environments (e.g. Virtual Breading Environments, Virtual Organisations etc.) and taking into account the several efforts that we have reviewed, we argue on shifting the attention towards assisting the end-users of collaboration in (semi)-automatic ways and developing new tools that can promote flexible recommendations for real-time corrective actions in ongoing collaborations.

Among these we also found efforts on patterns that use in some degree ontologies. S. Henninger proposed an ontology based (OWL) model for Usability Patterns [11]. Although this pattern meta-model is not designed for collaboration patterns it is significant work that proves the benefits of ontology based modelling of patterns and gives concrete usage scenarios for a pattern ontology. Biuk-Aghai et al. [28] proposes the use of an ontology in order to describe and map between different levels of information, expressed in the Information Pyramid of Virtual Collaboration. Moreover ontologies for virtual collaboration patterns are mainly used to communicate meaning, and to reuse and organize knowledge. The e-Ace project [29] proposed an ontology structure that implements a "collaboration stack". This ontology maps the various levels of pattern abstraction, ranging from abstract collaboration patterns to collaborative services and communication technologies. In that way it serves as a pattern hierarchy, allowing the automatic selection of lower-level patterns upon the selection of specific abstract patterns. Unified Activity Methodology has introduced an activity meta-model in the form of ontology [30]. According to this, an activity is represented as an association of properties and as relationships to other entities.

5. CONCLUSIONS

In this paper, we presented a pattern-based approach for supporting collaborative processes using ontologies. With this approach we aim to fuse process-centric with ad-hoc, knowledge-based collaboration in an effort to balance reusability and flexibility. Balancing flexibility and reusability is needed in real-life, large scale collaborations (e.g. collaboration inside VOs) in which software services and human actors are involved. The benefits of the proposed coupling of a pattern-based approach with ontologies include intelligently matching collaboration contexts and requirements to collaboration patterns, making intelligent inferences about applying patterns to solve problems at successive levels of abstraction and recommending either workflows or less structured actions as solutions to collaboration contexts.

Our future work includes the integration of the CPA tool in an Event Driven Architecture (EDA) environment where specific collaboration services (e.g., services supporting communication, collaboration and coordination) are available; these services generate events which provide triggers for the recommendation of CPats. We also plan to evaluate CPA by using it to support collaboration in real VOs from the pharmaceutical and manufacturing domains.

6. ACKNOWLEDGMENTS

This work has been partially funded by the European Commission, project SYNERGY (Supporting highlY-adaptive

Network Enterprise collaboration thRouGh semanticallY-enabled knowledge services), ICT No 63631. The authors would like to thank the project team for comments and suggestions.

7. REFERENCES

- Hlupic, V., Qureshi, S. 2003. A research model for collaborative value creation from intellectual capital, In: Twentififth International Conference of Information Technology Interfaces. Cavtat, Croatia.
- [2] Verginadis, Y., Apostolou, D., Papageorgiou, N., Mentzas, G. 2009. Collaboration Patterns in Event-Driven Environment for Virtual Organisations. Intelligent Event Processing – Association for the Advancement of Artificial Intelligence (AAAI), Spring Symposium, Stanford, USA.
- [3] Papageorgiou, N., Verginadis, Y., Apostolou, D., Mentzas, G. 2009. A Collaboration Patterns Model for Virtual Organisations. In PROVE'09, 10th IFIP Working Conference on Virtual Enterprises, Thessaloniki, GREECE, Springer, pp. 61-68.
- [4] Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P.,Stal, M. 1996. Pattern-oriented software architecture: a system of patterns, John Wiley & Sons, Inc. New York, NY, USA.
- [5] Alexander, C., Ishikawa, S., Silverstein, M. 1977. A Pattern Language. New York, Oxford University Press.
- [6] Gamma, E., Helm, R., Johnson, R., Vlissides, J. 1995. Design patterns: elements of reusable object-oriented software, Addison-Wesley Reading, MA.
- [7] Borchers, J. O. 2001. A pattern approach to interaction design." AI & Society 15(4): 359-376.
- [8] Gruber, T. R. 1993. A translation approach to portable ontology specifications. KNOWLEDGE ACQUISITION, 5, 199-199.
- [9] Verginadis, Y., Apostolou, D., Papageorgiou, N., Mentzas, G. 2009. An Architecture for Collaboration Patterns in Agile Event-Driven Environments. Fourth IEEE Workshop on Agile Cooperative Process-Aware Information Systems (ProGility 2009), Groningen (The Netherlands).
- [10] de Moor, A. 2006. Community Memory Activation with Collaboration Patterns. Proceedings of the 3rd International Community Informatics Conference (CIRN 2006) Prato Italy, pp. 1.
- [11] Henninger, S., Ashokkumar, P. 2006. Disseminating Usability Design Knowledge through Ontology-Based Pattern Languages. In Proceedings Semantic Web User Interaction Workshop, ISWC2006, Springer's LNCS
- [12] Stanford, U. 2008. Protege 3 Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine, http://protege.stanford.edu.
- [13] Sintek, M. Ontoviz tab. 2003. Visualizing Protege ontologies, 2003.
- [14] Parsia, B., Sirin, E. 2004. Pellet: An OWL DL Reasoner.
- [15] NAICS 2007. North American Industry Classification System, http://www.census.gov/eos/www/naics/.

- [16] Schall, D. Truong, H.L., Dustdar, S. 2008. Unifying Human and Software Services in Web-Scale Collaborations, IEEE Internet Computing, pp. 62-68.
- [17] Schümmer, T., 2002. Constructing a Groupware Pattern Language. Workshop on Socio-Technical Pattern Languages, part of CSCW
- [18] Adobe/Flex/AIR, available online at: http://www.adobe.com/products/air/
- [19] Flash Server Red5, available online at: http://www.red5.org/
- [20] ESPER engine, available online at: http://esper.codehaus.org
- [21] INTALIO workflow engine, available online at: http://www.intalio.com/
- [22] Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., Konig, D., Leymann, F., M"uller, R., Pfau, G., Pl"osser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., Zeller, M. 2007. WS-BPEL Extension for People (BPEL4People), Version 1.0
- [23] Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., K"onig, D., Leymann, F., M"uller, R., Pfau, G., Pl"osser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., Zeller, M. 2007. Web Services Human Task (WSHumanTask), Version 1.0
- [24] Briggs, R. O. 2003. Collaboration Engineering with ThinkLets to Pursue Sustained Success with Group Support Systems. Journal of Management Information Systems 19(4): 31-64
- [25] van der Aalst, W. M. P., ter Hofstede, A. H. M. 2005. YAWL: yet another workflow language. Information Systems 30(4): 245-275
- [26] Barros, A. P., M. Dumas, ter Hofstede, A. H. M.. 2005. Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. LNCS, pp. 302-318
- [27] Zhao, L., Macaulay, L., Adams, J., Verschueren, P. 2007. A pattern language for designing e-business architecture. The Journal of Systems & Software
- [28] Biuk-Aghai, R. P. 2003. Patterns of Virtual Collaboration. University of Technology, Sydney Faculty of Information Technology
- [29] eAce Project 2005. Available online at:
- [30] Moran, T. P. 2005. Unified Activity Management: Explicitly Representing Activity in Work-Support Systems. Workshop on Activity: From a Theoretical to a Computational Construct.