An Architecture for Collaboration Patterns in Agile Event-Driven Environments

Yiannis Verginadis, Dimitris Apostolou, Nikos Papageorgiou, Gregoris Mentzas
Institute of Communications and Computer Systems
National Technical University of Athens
Athens, Greece
{jverg, dapost, npapag, gmentzas}@mail.ntua.gr

Abstract— Collaboration in Virtual Organizations is usually informal, dynamic and ad-hoc. In this paper, we introduce Collaboration Patterns as a mechanism to facilitate knowledge-based collaborations. We also propose an event-driven architecture for modeling, recommending and enacting such Collaboration Patterns.

Keywords— collaboration pattern; agile event-driven environment; process aware

I. INTRODUCTION

In the modern competitive business environment the notion of collaboration involves the complex nature of vast number of processes that may span across organizational and geographical boundaries. Moreover businesses face challenges that the single enterprise model is not sufficient to cover, therefore, they establish virtual but legally consolidated schemes for collaborating; such schemes are the virtual organizations (VOs) [1].

The environment in which virtual collaborative environments operate is subject to continuous changes because partner participation may be dynamic, information and communication technology systems used may be changing and business goals may be evolving. In such changing circumstances there is a need for adapting the ways of collaboration to reflect the current conditions. A way to address this challenge is by identifying and appropriately reusing Collaboration Patterns (CPats). CPats are segments of collaboration work, which can be identified as recurring and can be reused. The reuse of CPats can constitute an advantage in agile collaborative environments such as VOs, where there is an increased need for modelling, executing, monitoring and supporting the dynamic nature of collaborations.

The overall aim of using CPats in the VO field is to enable adaptivity of collaborations with respect to changing circumstances. To achieve adaptivity, a mechanism for detecting and appropriately responding to changing conditions is needed. Event-Driven Architecture (EDA) can provide such a mechanism. EDA is a software architecture approach for promoting the production, detection, consumption of, and reaction to events [2].

In this paper, we present our work in progress regarding the combination of an event driven architecture and collaboration pattern services for coping with the critical challenges that collaborative environments pose. In section 2, we present related work, while in section 3, we talk on patterns and CPats. We continue in section 4 by presenting an architectural proposition for demonstrating the CPats’ applicability. Finally, this paper concludes with a discussion regarding future work on this effort.

II. RELATED WORK

In the domain of inter-organizational collaboration, standards such as RosettaNet (www.rosettanet.org), XCBL (www.xcbl.org) and ebXML (www.ebxml.org) define, based on XML, message guidelines, business processes interface and implementation frameworks for the sharing of business documents. Moreover, they facilitate the development of coordination protocols of the possible process scenarios from the intersection of the partners’ process specifications. There exist also Web service-specific standards for process composition, orchestration and choreography, such as WSCI and WS-CDL Patterns have been mainly examined in inter-organizational collaboration to support control flow [3], data flow [4], resources [5] and service interaction [6], facilitating IT-based collaborations without taking into account the support of human collaborations.

The focus of our work, in contrast to the aforementioned ones, is on using patterns to support informal, loosely defined knowledge-based collaboration (see also [7]) rather than on agreeing on shared business content and coordination protocols. Our goal is to address the specificity frontier of collaboration support systems [8], i.e. to support a range of collaboration types, from fixed to ad-hoc. In this respect, we extend research on activity patterns [9] and propose CPats as a type of design patterns applicable to event-based systems [10]. Regarding our architectural proposition, we extend previous research on event-based architectures for cross-organizational coordination [11], as well as efforts that adopt patterns for knowledge workflow management [12].

III. COLLABORATION PATTERNS

A. The Need for Patterns

When experts work on a particular problem, it is unusual for them to tackle it by inventing a new solution that is completely distinct from existing ones. 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 behavior’ is a natural way of coping with any kind of problem or social interaction [13].
For many people, the word pattern has appeared almost entirely due to the work of Christopher Alexander, a professor of architecture at the University of California at Berkeley. His pattern language book [14], is seen as the prototype to patterns books in software. During this first use of patterns in architecture, he defined a pattern as a “morphological law that explains how to design an artifact in order to solve a problem in a specific context”.

This concept has also been adopted in the context of software engineering, in order to capture best practices for creating software and using common concepts to solve recurring problems. The first notable publication in this area was the book "Design Patterns: Elements of Reusable Object-Oriented Software" by the so-called Gang of Four [15], that advanced the popularity of patterns in computer science.

B. What is a Collaboration Pattern?

In general, the participants of a collaboration process need to be collocated either physically or virtually in order to make a decision or work together. During such collaborations, recurring segments of actions can be identified and introduced as CPats for future use.

Towards the direction of having a comprehensive description of what a CPat is, we propose the following definition:

“A collaboration pattern (CPat) is a prescription which addresses a collaborative problem that may occur repeatedly in the environment. 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. The CPat can 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.”

For a solid and unambiguous description of a CPat, we need attributes that present the description of the problem and its solution that the CPat can address in a specific context and under specific collaboration circumstances. This description can be presented as a model [16] for describing CPats and involve several simple elements (e.g. Pattern Name, Pattern Id, Duration, Application Area, etc.) and a number of complex ones (Pre/Post-Conditions, Triggers, Solution etc.). Important elements of a CPat’s description regarding its detection and instantiation are the pre-conditions, that dictate the states and conditions that must be satisfied before the specific CPat can be considered applicable and the triggering parts of its execution. In an event-driven architecture, simple or complex events can trigger the execution of a specific collaboration pattern satisfying the need for reactivity and attention amplification in the environment of collaborative enterprises. The Solution element of the CPat’s description comprises of prescriptions of solutions to the designated problem and usually involves actions, workflows or even collaborative tools.

Regarding the life-cycle of a CPat, it involves the familiar path of: Creating, Translating, Simulating, Executing and Evaluating a CPat. The first step of the CPat lifecycle results in the CPat’s representation in both a graphical notation such as BPMN which gives an overview of “possible” ways of collaboration (processes, tasks, actions and tools) that can be altered even at run-time and in a machine processable language such as XML. The translation phase involves the production of a set of rules that capture the CPat pre-conditions and triggering mechanisms and also the creation of a set of mappings to executable files (e.g. BPEL) that can implement the described CPat solution. Based on a simulation mechanism, CPats should be simulated and validated against a number of restrictions that may have been imposed. As CPat execution we define first the retrieval and instantiation of the appropriate CPat for the collaboration situation (e.g. preconditions, triggering mechanisms, solution) and second the implementation of the CPat’s solution by the appropriate “execution engine” (e.g., workflow engine, groupware tool, human tasks, etc.). The last step of the CPat lifecycle which is the evaluation may lead to the creation of a new CPat or the refinement of the existing one.

C. A Collaboration Pattern Example

In this section, we present a CPat that deals with the case in which a VO member must be removed due to poor performance (Fig. 1). Specifically, we describe this CPat focusing on some of the most important aspects of the CPat model mentioned in the previous section.

This CPat can be triggered when a specific pattern of simple events occurs and a specific set of facts hold as pre-conditions. In our example, when a VO member does not participate in the VO for X weeks (this is a complex event comprised of the following events: (i). VO member does not enter the VO forum, (ii). VO member does not appear in meetings, (iii). VO member does not reply to VO coordinator’s mails) and if the VO coordinator is allowed to decide a partner removal (i.e. fact related to the collaboration) then the specific CPat is recommended to the appropriate user for execution.

In table 1, we present ECA pseudocode snippets that correspond to the triggering mechanism of our CPat example. The IF part includes zero or more gating requirements (a “?” indicates a variable).
TABLE I. TRIGGERING MECHANISM

<table>
<thead>
<tr>
<th>ECA</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHEN</td>
<td>Doesn’t_participate_in_VO (?VOmember, ?Period) OR VO_Coordinator_decides_for_partner_removal (?VOmember)</td>
</tr>
<tr>
<td>IF</td>
<td>canDecideForPartnerRemoval (?VOcoordinator) AND performance(?VO member)=&quot;poor&quot;</td>
</tr>
<tr>
<td>THEN</td>
<td>Withdrawal of VO Member CPat (?VOmember)</td>
</tr>
</tbody>
</table>

The proposed CPat solution involves a set of actions to be undertaken by the VO coordinator and the VO members. A diagrammatic description of the CPat solution is given in Fig. 3. We note that although we use a strict formalism in order to graphically present (BPMN) the CPat’s solution, we perceive it only as a “possible” sequence of actions that are bound to be changed by the user that will accept the CPat recommendation.

IV. ARCHITECTURE FOR AGILE PROCESS AWARE COLLABORATIONS

This proposed architecture articulates a conceptual view of a system that supports the description, instantiation and execution of CPats during the life cycle of VOs. In order to demonstrate the applicability of CPats we propose the combination of the following software components (Fig. 2):

- An Event Service Bus (ESB) which acts as a lightweight, ubiquitous integration backbone through which events flow through intelligent routing mechanisms. It acts as a bridge between the several event sensors that perceive events from an event cloud and the complex event processing engine.
- A Complex Event Processing Engine (CEP Engine) which has the capability to process simple events (e.g., coming from groupware applications), combine them and produce complex constructs of events that may trigger CPats. An Ad-hoc Workflow Engine which is responsible for implementing the CPat’s solution and produce events that can be used by the CPA for forming an overview of the state of execution/collaboration. This component enables adaptivity by allowing the modification of the actions described by the CPat the moment the later is instantiated (e.g., modify a sequence of actions from sequential to parallel, delete an action, etc.).

A Collaboration Pattern Processing Layer (CPP Layer) which validates CPats according to a set of predefined restrictions, instantiates CPats and generates reaction rules bounded to the collaboration context that depict for which pre-conditions and triggers CPats can be executed. Triggers may involve complex events whose description is communicated to the CEP engine in order for them to be detected in run-time. It also manages execution and access rights related to CPats. When the execution of the CPat solution starts, it generates a set of expected post-conditions, exceptions and duration time outs which are removed upon CPat termination.

A Collaboration Pattern Editor (CPat Editor) which defines, edits, searches and simulates CPats. It produces abstract CPat which can be instantiated and adapted during run-time.

A Collaboration Pattern Assistant (CPat Assistant) which, upon notification from CEP for complex events, uses facts from Collaboration Knowledge Base to evaluate the preconditions specified in rules and derives recommendations. If the CPat Assistant user accepts the recommendation for a CPat, the CPat Assistant instructs through the CPP layer the execution of the CPat solution in the Ad-hoc Workflow Engine.

The proposed rule-based approach for recommending CPats is based on event-condition-action (ECA) rules [17]. This type of rules is understood as follows: whenever a (complex) event occurs and conditions hold, the actions of all matching rules are executed. Simple events arrive at the CEP, which maintains (partially detected) complex events in its pattern base. Event patterns, remain in the store until they reduce to true (occurred) or false (impossible). As soon as an event pattern reduces to true, CEP notifies CPA about the corresponding complex event occurrence. CPA will check for the preconditions of the CPat by searching the collaboration knowledge base. Upon preconditions validation, CPA derives recommendations by matching the current collaboration state to one or more CPats stored in the CPat base. The actual instruction of the ad-hoc workflow engine takes place through the acceptance of the CPA recommendation.

In the persistence level three information sources are required. Firstly, the collaboration knowledge base which contains domain models as well as facts about the collaboration that may change as the collaboration proceeds. It provides information about the state of collaborators and the state of collaborative work, while it aggregates events and their consequences for the ongoing collaborations. Secondly, a collaboration pattern base and an event pattern base are needed for storing CPats and event patterns respectively.

This proposed architecture is being currently implemented in a prototype system for which we use open source solutions that can be customized and combined for our purposes. Specifically, we are using the PeaTALS [18] enterprise service bus, the ESPER complex event processing engine [19] for processing event streams and the MAESTRO workflow engine [20] for handling ad hoc workflows.
V. CONCLUSIONS AND FUTURE WORK

In this paper we discussed the introduction of CPats in event driven architectures in order to facilitate collaboration within VOs. Collaboration in VOs is characterized as dynamic and ad-hoc; it is often not easy to explicitly define in advance the exact form of collaboration. VOs have increased needs for modeling, executing, monitoring and supporting the dynamic nature of collaborations, which can be favored by the combined strength of CPats and events acting as patterns for reoccurring segments of collaborative work and triggering mechanisms for such patterns, respectively. We believe that this approach can cope with the critical challenges that a VO environment poses. Further work will be carried out in terms of implementing this combination of event driven architecture with CPats and validating it across real case scenarios in VOs lifecycles.

ACKNOWLEDGMENT

This work has been funded by the European Commission regarding the strep project SYNERGY (ICT No 63631).

REFERENCES

[18] PEtALS, Open Source ESB, available online at: http://petals.ow2.org/

Figure 3. Collaboration pattern solution.