Collaboration Pattern Modeling in Support of Norm Specification, Monitoring, and Preservation*

Christoph Dorn[1]

1 Vienna University of Technology, Distributed Systems Groups
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
dorn@infosys.tuwien.ac.at

Abstract

Collaboration-intensive environments call for technical systems that permit flexible user interactions. Rigid workflows are no suitable collaboration paradigm. As users apply various patterns such as shared artifact, social networks, client/principal, or publish/subscribe for interaction, their cooperative behavior becomes largely determined by norms. In this paper, we make the case for explicit modeling of collaboration patterns as the substrate for specifying, monitoring, and preserving norms. Describing collaboration patterns in the form of human-centric component and connector architecture views provides a means for reasoning on collaboration control, flexibility, and ultimately adaptability. We report on recent work targeting executable collaboration patterns and outline resulting synergies with norms.

1998 ACM Subject Classification D.2.11 Software Architectures

Keywords and phrases Collaboration Patterns, Human Architecture Description Language, Collaboration Execution Framework

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Today, various internet-based platforms exist that build upon the network effect to promote sustainable living, healthy lifestyles, or social inclusion. Generally described as Collective Awareness Platforms (CAP), these platforms range from shared gardening initiatives to food redistribution or volunteering for disaster relief. [1] User autonomy and pro-activeness are key collective intelligence characteristics. Rather than centrally controlling a user community, CAPs need to foster individual participants in volunteering for various personal reasons and to different degree of intensity. Norms gain an increasingly important role in such environments for determining and guiding, for example, sharing of limited resources, allocating jointly produced output, or collaborative decision making.

Contemporary platforms typically specify norm-related aspects in free-text form such as rules, best-practises, or work procedures. Participants, however, don't directly engage in explicit norm enactment but interact through collaboration mechanisms. Example mechanisms include email, Question/Answer forums, chat rooms, Wiki articles, Facebook activity streams, Twitter-based news dissemination and many more. These mechanisms, per se, are norm independent but affect to what extent we may specify, observe, restrict, and ultimately guide

* This work was partially supported by EU FP7 Smart-Society project, under Grant No. 600854.
[1] https://ec.europa.eu/digital-agenda/en/projects-and-initiatives-driving-sustainable-behaviours

norm compliance, respectively violation. A Wiki article, for example, enables monitoring article changes and protecting it in case of vandalism.

Available collaboration mechanisms and their specific configuration become ever more important as a CAP's community grows. At the beginning, communities are small enough to manually maintain trust and cooperation. Excessive control mechanisms such as chat room moderators, article patrols, update voting, or workflow engines guaranteeing task execution are perceived as signs of general distrust, collaboration barriers, and ultimately impede community growth. In contrast, a grown community requires such control mechanism for enforcing norms, respectively for deterring freeloading behavior. Beyond a certain size, norm observation and sanctioning is beyond the capability of individual members. Instead, collaboration mechanism adaptation and/or additional mechanism deployment becomes necessary. In short, rigid collaboration mechanisms stifle collaborative efforts as they won't match the evolving community's need for norm enforcement, respectively guidance.

Collaboration mechanism adaptation and dynamic deployment is non-trivial due to the underlying collaboration patterns' idiosyncracies. Patterns such as single messages, information streams, shared artifacts, social network, or publish-subscribe exhibit significantly diverse properties in terms of control, participant decoupling, collaboration state handling, and change pre-conditions [5].

Before we can link norm specifications (and any sophisticated monitoring and adaptation based on those specifications) to CAPs, we need (i) a good understanding of collaboration pattern adaptability and (ii) a suitable language for describing patterns abstracted from concrete technology implementations (e.g., Facebook, Twitter, Wikipedia, respectively custom-made platforms). We believe, a mapping between norms and patterns holds much potential for reasoning on norm compliance, determining need for additional information sources (i.e., patterns) for norm monitoring, and warnings on what user actions may lead to violations. Abstracting from concrete technology platforms enables reuse of norm-centric analysis and adaptation planning algorithms.

In this paper, we first outline a motivating scenario that highlights the interplay of norms and collaboration patterns (Sec. 2). We subsequently provide an overview on collaboration pattern adaptability (Sec. 3), introduce pattern modeling with the human Architecture Description Language (Sec. 4), and report on recent progress on pattern specification, execution, and monitoring (Sec. 5). We conclude with a discussion on related work and joint interdisciplinary research steps.

2 Motivating Scenario

The 2010 Haiti earthquake gave rise to the first form of virtual disaster management. Remote volunteers assisted in information cleansing and activity coordination through general purpose social media [10]. Future CAPs for collaborative disaster management will have to enable dynamically integrating activities from volunteers, NGOs, and various local authorities at a pace that custom-made platforms cannot keep up with and where general purpose social media platforms provide insufficient interaction patterns for. Imagine a platform (Fig. 1) that enables raw information collection from volunteers, NGO members, rescue personnel at the disaster site, streaming video feeds to volunteers for content analysis, having back-office operators coordination information flow and activities 24/7 in shifts, seamlessly integrating response team deployment via dispatchers.

Norms play a key role in governing the (expected) behavior of the various participants. On-site observers have extensive freedom what to report and when to report. Depending on whether they are volunteers, NGO members, or rescue personnel, different expectations in terms of timeliness and accuracy apply. While norms allow the formulation of such expectations, they cannot be observed or managed (i) without a mapping to the various applied collaboration patterns, or (ii) without a clear understanding what control and flexibility these patterns provide in terms of adaptability. [2]

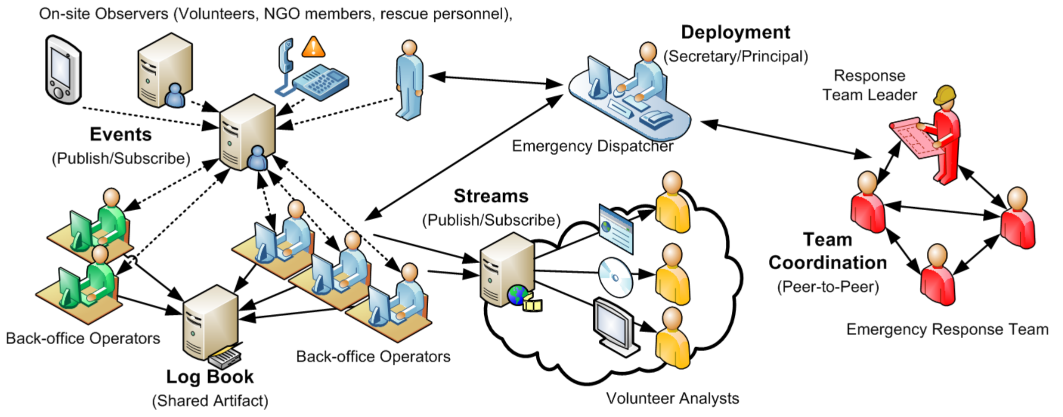

Figure 1 Motivating Scenario: Collective Disaster Management

3 Collaboration Patterns Adaptability

Malone and Crowston highlighted the existence of similar coordination needs such as information flow, control flow, resource access, task execution in Economics, Organizational Theory, and Computer Science [6]. Applied to human collaboration we can identify (without claim for completeness) a set of seven fundamental patterns [5]: *Shared Artifact*, *Publish/Subscribe*, *Master/Worker*, *Social Network*, *Workflow*, *Secretary/Principal*, and *Organizational Control*.

These patterns have significant similarities to software architectural styles [11]. Adaptability analysis of software architectural styles such as blackboard, client/server, peer-to-peer, etc, hence, serves as the basis for collaboration pattern analysis. Specifically, we apply the BASE methodology [12]: Behavior, Asynchrony, State, and Execution. For each pattern, we elaborate what pattern elements can and cannot (easily) change (behavior), the duration of enacting a change and it's effect on the system (asynchrony), where collaboration awareness and know-how are captured (state), and under which conditions a change may be enacted (execution). Given the users' autonomy, we additionally observe which pattern elements may induce the changes and hence gain insights into whether norms are preserved through proactively enforcement (e.g., a on-site volunteer cannot push messages to a back-office operator unless he follows her), re-actively mitigation (e.g., protecting a Wiki article documenting the disaster), or whether sanctioning mechanisms need to be in place to induce cooperative behavior, respectively deter malicious behavior (e.g., up/down voting volunteering contributions instead of blocking the volunteer).

Table 1 provides a summary of pattern adaptability. While it doesn't provide detailed recommendations it nevertheless allows insights into what collaboration patterns may be appropriate for managing participants in disaster management.

[2] This paper focuses on the latter challenge, while the former challenge, so we believe, would serve as an interesting discussion topic during the seminar.

Pattern	Behavior	Asynchrony	State	Execution
Shared Artifact	Collaborators join/leave and react to artifact changes; artifact restructuring	participant switching and artifact restructuring occurs locally; artifact remains inaccessible during restructuring	State captured in artifact (no collaborator internal state assumed)	Potential delay until editing has completed
Publish/ Subscribe	Publisher and subscribers join/leave (stable connector); switch message topic production/ consumption	Messaging connector buffers messages	No shared state among participants; subscription state maintained by connector	no constraints unless adaptation involves messaging connector
Master/ Worker	Add/remove clients, workers (and masters); change Job-to-Worker assignment	Newly posted task, repetition, reassignment, replication wait for available, suitable worker; task update requires reposting	State limited to task status and job assignment/result	Wait for tasks completion; task cancelation incurs compensation costs; failing workers incur (reputation) penalty
Social Network	Members join/leave; establish/remove links arbitrarily	Member and link changes occur constantly	Replicated across members, partially maintained by super-peers	No constraints as requests are expected to fail
Workflow	Add/remove workers; update control/data flow specification	Workflow connector buffers tasks, distribution to workers per workflow instance; changes of workflow specification typically available in future instance	No shared state among Workers, all state with workflow connector	Wait for task completion for worker adaptation or task repetition; wait for next workflow instance or repeat workflow for task specification update
Secretary/ Principal	Add/remove client/secretaries/ principals; change allocation clients to secretary, secretary to principal	Processing delays due to redirecting, buffering, rejecting/retrying requests	Requests carry state; ideally stateless secretary and principal (but rarely the case in reality)	Complete processing all requests before change or redirect; buffer-/reject new requests
Organizational Control	Add/remove employees; restructure organization hierarchy	Changes delay information flow from/to linked sub-hierarchy to remaining organization	No shared state among employees; state is employee internal	Wait for completing all downward requests and upward-replies before changing employee; buffering of notifications

Table 1 Overview of BASE properties for each collaboration pattern [5].

4 The human Architecture Description Language

Our adaptability analysis highlighted the importance of dedicated elements which facilitate and manage interactions. Managers, team leaders, secretaries, discussion moderators, or emergency dispatchers exemplify real world collaboration roles that have primarily coordination duties. Software architecture provides fundamental concepts for distinguishing among work and coordination roles. The basic elements in a software architecture are components and connectors [11]. At any given level of abstraction, components are the loci of computation and data management whereas connectors coordinate the interactions between components.

Along these line, our human Architecture Description Language (hADL) [4] provides a collaboration-centric component and connector view. A hADL model describes the collaboration structure in terms of the interacting user roles and their available interaction mechanisms. hADL distinguishes between *HumanComponents* and *CollaborationConnectors* to emphasize the difference between the primary collaborating users and non-essential, replaceable users that coordinate the collaboration. A CollaborationConnector is thus responsible for the efficient and effective interaction among HumanComponents. It thereby may cover the full automation spectrum: from purely human, to software-assisted, to purely software implemented. Users employ diverse means of interaction that range from emails, to chat rooms, shared wiki pages, and Q&A forums, to vote collection. These means implement vastly different interaction semantics: a message is sent and received, a shared artifact is edited, a vote can be cast. *CollaborationObjects* abstract from concrete interaction tools and capture the semantic differences in subtypes; e.g., *Message*, *Stream*, or *SharedArtifact*. The hADL model excerpt in Figure 2 depicts *On-Site Observer*, *Analyst* and *Operator* human components. The pure software-based *VideoPubSub* collaboration connector takes video streams from observers and delivers them according to *Subscriptions* to volunteering analysts and operators. Subscriptions simultaneously serve as shared artifact for promoting awareness among operators which video sources are currently under observation. Analysts and operators stream relevant videos onto a shared *Wall Screen*.

Actions specify what capabilities a component or connector requires to fulfill his/her role, e.g., receive a stream or reset a shared artifact. Complementary, *Actions* on CollaborationObject determine the offered capabilities. To this end, actions distinguish between *Create*, *Read*, *Update*, and *Delete* (CRUD) privileges. Action cardinalities further specify the upper and lower boundaries on the number of collaborators which may simultaneously have acquired the action's capabilities. A Pub|SubStream's *out* action, for example, may demand at least one component or connector having receiving privileges when exhibiting an action cardinality of *(1..*)*. Ultimately, *Collaboration Links* connect HumanComponents and CollaborationConnectors actions to CollaborationObjects actions, thus wiring up a particular collaboration structure.

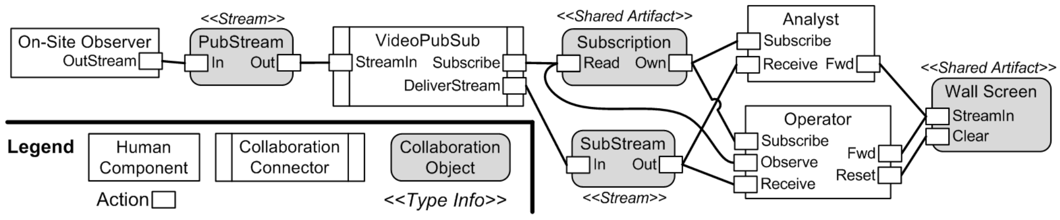

Figure 2 hADL core model instance excerpt.

5 Executable hADL Models in Support of Norm Operationalization

Avoiding norm violation through limiting users' available (inter)actions is often neither feasible nor desirable. Alternatives include dynamically adapting the underlying collaboration structures and/or providing recommendations where there is too little control over the collaboration elements.

To this end, our current work focuses on executable hADL models. We aim for dynamically instantiating collaboration objects, attach desirable collaboration connectors, and linking the newly created collaboration structure to existing (and additional) human components. Based on the hADL core model (see previous section), we introduce the hADL runtime and hADL executable model. The latter captures the details required for interacting with collaboration objects, collaboration connector, and human components via so called *Surrogates*. Figure 3 outlines the interplay of the various hADL schemata, model instances, runtime framework software components, and surrogates in our early proof-of-concept implementation.

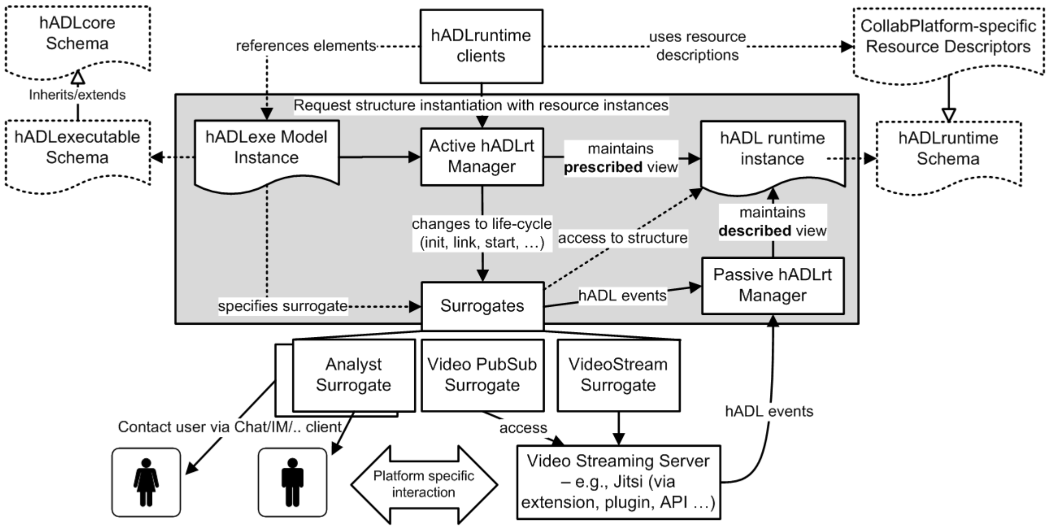

Figure 3 hADL Runtime Framework for executing hADL models

A hADL runtime client references all desired element types from a hADL executable model and the resource instances (i.e., humans, software, hardware) that should populate these elements (e.g., LinkedIn user Alice as human component *Analyst*). The active hADLrt Manager parses the input hADL specification and creates surrogates for each resource instance. It maintains the desired (i.e., *prescribed*) collaboration topology as a hADL runtime instance. The Passive hADLrt Manager maintains the respective *described* view: to what extent the collaboration update requests and recommendations have successfully manifested as changes in the real world. Surrogates are life-cycle managers for hADL elements. They receive requests for acquiring, linking, starting, etc of collaboration objects, collaboration connectors, and human components. Whether they enforce these request (e.g., automatically deploying a video stream) or merely recommend actions (e.g., asking a volunteer to subscribe to a stream) depends on the underlying platform's capabilities. To this end, a collaboration object surrogate communicates with the underlying technical collaboration platform (via it's API, hADL-specific extension point, or plug-in; the technical details are out of scope) to manipulate the respective collaboration object (e.g., set up a video stream). A surrogate associated with a human component, on the other hand, utilizes the collaboration platform's notification mechanism or third-party communication channels for informing the respective user about his/her collaboration involvement (changes).

The hADL runtime framework provides the basis for sophisticated adaptation actions. A norm preservation component may assume the role of a hADLruntime client and requests the deployment of appropriate connectors (recall that connectors coordinate and facilitate collaboration). Adaptation plans may involve switching from a permissive wall screen access connector to a restrictive one that introduces a new human component type for checking and confirming changes.

Note that our framework doesn't assume full control over the participants' involvement. Available collaboration capabilities may allow external participants to dynamically join and leave (similar to Wikipedia editors). While the hADL runtime manager typically has no control over those participants, sensor logic in surrogates and collaboration platforms provide details on their existence and behavior.

6 Discussion and Related Work

Our current work has yet to be integrated with research efforts on norms. Given the users' autonomy and consequently demanded action flexibility in CAPs, norms and collaboration patterns emerge as highly complementary, synergistic approaches. Interaction patterns may serve as the substrate for specifying, monitoring, analyzing norm compliance, as well as providing the mechanism for executing adaptation plans for avoiding, mitigating, or sanctioning norm violations. We make no claims that collaboration patterns are a suitable substrate for norms in any domain. Collaboration-intensive environments — the very focus of our research — however, typically exhibit a diverse mix of interaction mechanisms that strongly rely on cooperating (i.e., norm-driven) user behavior.

Traditional software-engineering approaches are insufficient for building CAPs. Existing goal-oriented requirements specification techniques such as i* [7], CSRML [13], or URN [1] can't distinguish between work executing and work coordinating user roles. Modeling approaches for context-aware or web-based applications [3, 8] are unprepared for expressing the extent of control and flexibility that participants enjoy. Existing message-based norm protocols [9, 2] fall short on similar aspects. They are cumbersome to apply to interactions emerging from shared artifact, publish/subscribe, or secretary/principal-based collaboration patterns.

7 Conclusion and Future work

We have introduced collaboration patterns modeling as a means to designing better collaboration-intensive applications. Analyzing adaptability give us insights into the pros and cons of various patterns in terms of resulting control over and flexibility of participants. Our human Architecture Description Language promotes separation of work executing and work coordinating roles, subsequently fostering adaptation opportunities. We believe that collaboration patterns constitute a suitable abstraction layer from the technical platform which allows for simplified norm specification, observation, reasoning, and ultimately norm-driven collaboration adaptation.

We expect the Dagstuhl seminar to offer exciting discussions on the interplay of norms and collaboration patterns: on the one hand, producing insights how research on norms may benefit from collaboration pattern modeling and, on the other hand, how our research efforts on executable hADL models may benefit from norms as a mechanism for driving adaptation. Specific questions center on what type of norms are appropriately supported by collaboration patterns, what information and events does norm management require beyond those provided by collaboration structure monitoring, and how may collaborators define and adapt the norms that should govern/support their very same collaboration.

Acknowledgements I would like to thank Amit Chopra for insightful discussions on the interplay of norms and collaboration patterns.

References

1 Daniel Amyot. Introduction to the user requirements notation: Learning by example. *Comput. Netw.*, 42(3):285–301, June 2003.

2 Alexander Artikis, Marek Sergot, and Jeremy Pitt. Specifying norm-governed computational societies. *ACM Trans. Comput. Logic*, 10(1):1:1–1:42, January 2009.

3 Stefano Ceri, Florian Daniel, Federico M. Facca, and Maristella Matera. Model-driven engineering of active context-awareness. *World Wide Web*, 10:387–413, December 2007.

4 Christoph Dorn and Richard N. Taylor. Architecture-driven modeling of adaptive collaboration structures in large-scale social web applications. In Xiaoyang Sean Wang, Isabel F. Cruz, Alex Delis, and Guangyan Huang, editors, *WISE*, volume 7651 of *Lecture Notes in Computer Science*, pages 143–156. Springer, 2012.

5 Christoph Dorn and Richard N. Taylor. Analyzing runtime adaptability of collaboration patterns. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a, 2014.

6 Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26:87–119, March 1994.

7 John Mylopoulos, Lawrence Chung, and Eric Yu. From object-oriented to goal-oriented requirements analysis. *Commun. ACM*, 42(1):31–37, January 1999.

8 Wieland Schwinger, Werner Retschitzegger, Andrea Schauerhuber, Gerti Kappel, Manuel Wimmer, Birgit Pröll, Cristina Cachero Castro, Sven Casteleyn, Olga De Troyer, Piero Fraternali, and et al. A survey on web modeling approaches for ubiquitous web applications. *International Journal of Web Information Systems*, 4(3):234–305, 2008.

9 Munindar P. Singh. Norms as a basis for governing sociotechnical systems. *ACM Trans. Intell. Syst. Technol.*, 5(1):21:1–21:23, January 2014.

10 Kate Starbird and Leysia Palen. Working and sustaining the virtual "disaster desk". In Amy Bruckman, Scott Counts, Cliff Lampe, and Loren G. Terveen, editors, *Computer Supported Cooperative Work, CSCW 2013, San Antonio, TX, USA, February 23-27, 2013*, pages 491–502. ACM, 2013.

11 Richard N Taylor, Nenad Medvidovic, and Eric M Dashofy. *Software architecture: foundations, theory, and practice.* Wiley Publishing, 2009.

12 Richard N. Taylor, Nenad Medvidovic, and Peyman Oreizy. Architectural styles for runtime software adaptation. In *WICSA/ECSA*, pages 171–180, 2009.

13 Miguel A. Teruel, Elena Navarro, Víctor López-Jaquero, Francisco Montero, and Pascual González. Csrml: A goal-oriented approach to model requirements for collaborative systems. In *ER*, pages 33–46, 2011.