A Framework for Model-Driven Execution of Collaboration Structures

Christoph Mayr-Dorn(✉) and Schahram Dustdar

Distributed Systems Group, TU Wien, 1040 Vienna, Austria
{mayr-dorn,dustdar}@dsg.tuwien.ac.at

Abstract. Human interaction-intensive process environments need collaboration support beyond traditional BPM approaches. Process primitives are ill suited to model and execute collaborations for shared artifact editing, chatting, or voting. To this end, this paper introduces a framework for specifying and executing such collaboration structures. The framework explicitly supports the required human autonomy in shaping the collaboration structure. We demonstrate the application of our framework to an exemplary collaboration-intensive hiring process.

Keywords: human Architecture Description Language · Collaboration patterns · Collaboration configuration · Scripting collaborations

1 Introduction

Medical diagnosis, paper authoring, and peer reviewing are examples of collabora-tion-intensive tasks. Such tasks increasingly require multiple participants who benefit more from dedicated collaboration support than from rigid control and data flow specification. Collaboration support ranges across distinct forms and patterns [11] such as *Shared Artifact*, *Social Network*, *Secretary/Principal*, *Master/Worker*, or *Publish/Subscribe*. Contemporary process technology is ill equipped to provide such collaboration support in a general manner.

Business Process Management (BPM) approaches traditionally assume a single executing entity per task or activity. In the rare cases where multiple human process participants work on a joint task [12,20,21], process specifications per se contain no details with respect to the applicable communication, coordination, or collaboration structures. The core question we address in this paper is thus: how can we set-up and control flexible collaboration instances at runtime in support of joint task execution?

Our solution is a framework for model-driven execution of collaboration mechanisms. A collaboration model specifies an arbitrary combination of collaboration mechanisms such as shared artifacts, messages, streams, requests, and the corresponding user roles expressed in the human Architecture Description Language (hADL) [9]. At runtime, a client (i.e., a process) requests instantiation of a hADL model with actual users and maintains control over the collaboration instance via our framework.

S. Nurcan et al. (Eds.): CAiSE 2016, LNCS 9694, pp. 18–32, 2016.
DOI: 10.1007/978-3-319-39696-5_2

Our approach is complementary to existing process modeling and execution techniques. We don't need to awkwardly model collaboration aspects in terms of fine-grained task, control flow, or data flow primitives. Instead, we specify how a process obtains control over who, when, and how to involve particular users in a particular collaboration.

The evaluation use case demonstrates how our proof-of-concept framework may facilitate the collaboration in multi-participant tasks. Our approach thus provides processes along the specificity frontier [2]—from rigorously defined workflows to ad-hoc activities—a novel capability for configuring collaborations depending on process context: from automatically wiring up process participants and collaboration objects to providing collaboration guidance.

The remainder of this paper is structured as follows. Section 2 motivates our work based on a running scenario. We provide necessary background information in Sect. 3. Section 4 outlines the architecture, models, and internal workings of our framework. Section 5 demonstrates the application of our framework to a use case from the motivating scenario. We discuss related work in Sect. 6 before concluding this paper with a summary and outlook on future work in Sect. 7.

2 Motivating Scenario

Assume a collaborative employment process for a vacant post-doc position at a university department. The department is interested in obtaining consensus on the set of candidates invited for interviews and aims at executing the decision process in a transparent manner. The hiring committee establishes a set of criteria against which to evaluate the candidates. Each application is assigned to a team of two department members for preparing a detailed assessment report. All department members may give comments on any applicant such as whether they know them from conferences, co-authoring, etc.

All assessment reports are discussed by the hiring committee. Committee members are expected to prepare by reading through the reports prior to the meeting. The university's minority awareness officer inspects every assessment for ensuring that evaluations are free from bias and that a sufficiently diverse candidate set is considered for interviewing. When supported by a traditional process-centric system without integrated collaboration support, such a process very probably causes awkward handling of feedback into assessments, participants lacking process awareness and thus missing out on discussions or working on out-dated information, as well as delays due to limited potential for parallel work.

There is no single mechanism for collaboration that would fit the overall process. We exemplify the benefit of introducing shared artifacts (here documents that allow synchronous editing and commenting) as well as communication streams (here chat rooms) for discussions (Fig. 1). Shared artifacts primarily enable parallel work while limiting the potential for write conflicts and access to out-dated information. Chat rooms provide a well-known, well-scoped mechanism for discussing, enabling late participants to quickly catch up with

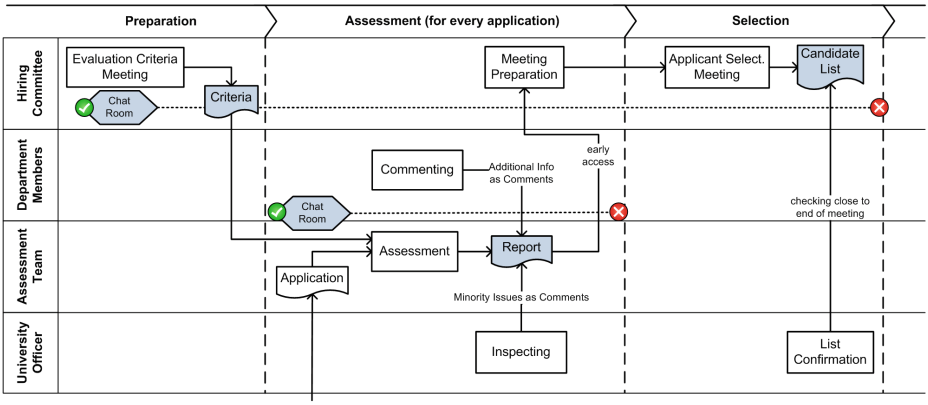

Fig. 1. Excerpt of a collaboration-intensive hiring process applying shared documents (shaded) and chat rooms (with dashed, horizontal life-lines). Process language specific details are omitted on purpose in order to abstract from integrations details and focus on the collaboration aspects and their potential impact instead.

the current state of the collaboration. The minority awareness officer may start early inspecting the reports without waiting for their finalization thus avoiding an overload on the assessment due date. Additionally, and more importantly, rather than escalating biased assessments after the deadline, any such concerns can be swiftly dealt with through timely feedback on a continuous basis. Similarly, the hiring committee can access the assessment reports early and just need to read-up on any last changes after the deadline (ultimately reducing the time needed to prepare for the application selection meeting). Realizing such a scenario requires a dedicated framework for managing the collaboration structure.

3 The human Architecture Description Language

We provide a brief introduction to hADL [9] as our approach makes heavy use of it. hADL provides a collaboration-centric equivalent of a software architecture "component & connector" view. A hADL model describes a collaboration structure in terms of interacting user roles and their available collaboration mechanisms. Figure 2 provides the hADL meta model (*elements in italics*). Figure 3 depicts the hADL model for the collaboration-intensive aspects of our motivating scenario (`elements in teletype`). Note that hADL's canonical representation is provided as an XML schema, available for download among the supporting online material (SOM) at http://wp.me/P1xPeS-6L.

hADL distinguishes between *HumanComponent*s (e.g., `DocUser` and `ChatUser`) and *CollaborationConnector*s to emphasize the difference between the primary collaborating users and non-essential, replaceable users that facilitate the collaboration. Collaboration connectors are responsible for the efficient and effective interaction among human components, respectively ensuring desirable collaboration outcome.

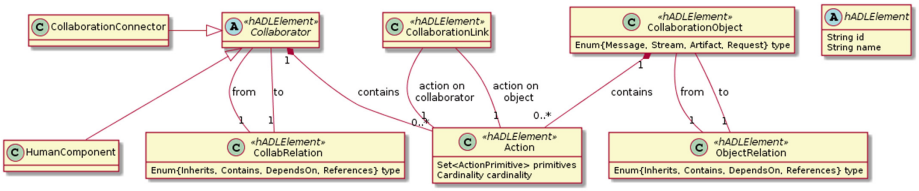

Fig. 2. Simplified and condensed hADL meta model displaying the elements relevant for the execution framework.

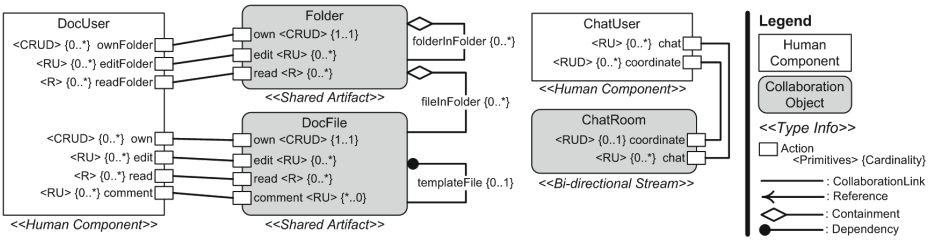

Fig. 3. hADL model excerpt that describes the main collaborators, collaboration objects and capabilities involved in the motivating scenario. DocUsers have either the capabilities to own, to edit, or to read Folders and Docfiles, or additionally to comment on the latter. ChatUsers have the ability to coordinate or to chat in a ChatRoom.

Humans employ diverse collaboration mechanisms that range from emails, shared wiki pages, social network activity streams, to Q&A forums and 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. hADL makes these differences explicit by means of *CollaborationObject*s. CollaborationObjects are first class modeling constructs which abstract from concrete interaction tools and capture the semantic differences in various subtypes such as *Message*, *Stream* (e.g., `ChatRoom`), or *SharedArtifact* (e.g., `DocFile`).

hADL Actions specify what capabilities a HumanComponent or CollaborationConnector requires for fulfilling their associated role, e.g., document authoring or providing comments. Complementary, actions on CollaborationObjects determine the offered capabilities. For example, editing a shared document (i.e., DocFile) requires the ability of performing a DocUser's `edit` action, while a ChatRoom offers the `coordinate` and `chat` actions. Additionally, hADL distinguishes among create (C), read (R), update (U), and delete (D) *primitives* to indicate the intended effect of an action. Further, action *cardinalities* specify the upper and lower boundaries on the number of collaborators which may simultaneously have obtained the action's capabilities. For example, exactly one user might own a document $\{1..1\}$, but many users might edit it $\{0..*\}$. *CollaborationLinks* subsequently connect actions that belong to HumanComponents or CollaborationConnectors to actions that belong to CollaborationObjects.

The human Architecture Description Language provides `CollabRelations` for modeling relations among HumanComponents and CollaborationConnectors as well as `ObjectRelations` among CollaborationObjects. For example, the specific `templateOf` relation may be applied for modeling that one `DocFile` *dependsOn* another `DocFile` which serves as template. Other relation types include *references* for specifying uni-directional relations between CollaborationObjects and *contains* for modeling hierarchical substructures.

Together, all these elements establish the blueprint of a collaboration structure. Note that hADL specifies the a-priori defined collaboration object types to be used at runtime (e.g., a `DocFile`), rather than their specific purpose within the (process) context (e.g., an assessment report).

4 The hADL Execution Framework

4.1 Architectural Overview

The primary purpose of model-driven collaboration execution is separating the specification of a collaboration structure (the what) from its realization on specific collaboration platforms (the how). This enables the hADL client—such as a process—to focus on the desired structure, the involved collaborators, and how the overall collaboration should evolve. Low-level details such as interacting with the various collaboration platforms through their APIs, maintaining collaboration state throughout the process' lifetime, or adaptation due to platform API changes remain hidden. Figure 4 depicts this separation of concerns. The main architectural elements and their duties are:

- the hADL client: requests instances of hADL elements to be created, re/wired, and released.
- the hADL Collaboration Linkage Connector (CLC): manages the collaboration structure, ensures valid client requests, and forwards those to surrogates for enactment.
- the Surrogates: translate hADL-centric client requests into invocations of the collaboration platforms.
- the hADL Runtime View: stores the current collaboration structure.

A hADL model describes the available element types (e.g., ChatUser, DocUser, ChatRoom, etc.) and their possible wiring but not an actual runtime topology involving actual humans. It's up to the **hADL client** to specify what instances of hADL elements from a particular hADL model it requires and how and when to wire them. To this end, the hADL client issues "acquisition" requests to the CLC which concrete users to involve in what collaboration-specific role (i.e., HumanComponent or CollaborationConnector) and what collaboration mechanism (i.e., CollaborationObjects) to utilize. A hADL client, for example, requests to involve user `Bob` as *Chat User*, and acquire a *Chat Room* with name `PreMeeting`. Here `Bob` and `PreMeeting` represent so-called *ResourceDescriptors* that describe identity and properties of users and collaboration mechanisms (see Fig. 5 middle). Once acquired, the hADL client determines

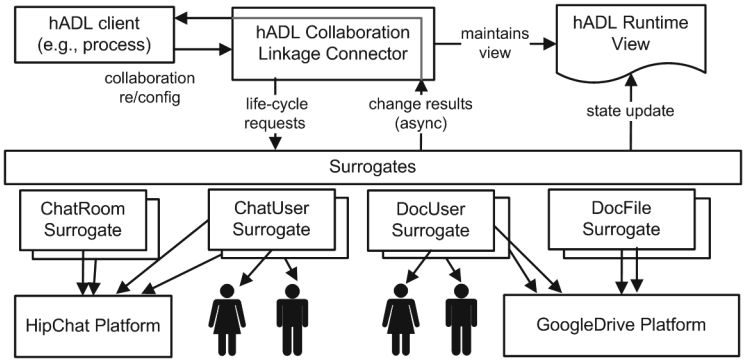

Fig. 4. Conceptual architecture of the hADL execution framework.

the wiring among instances of human component, collaboration connectors, and collaboration objects according to hADL actions, links, or relations. Wiring, gor example, ChatUser Bob to ChatRoom PreMeeting via action *coordinate*.

The **hADL Collaboration Linkage Connector** (CLC) takes the client's acquisition and rewiring requests and ensures they are valid according to the underlying hADL model. Its main purpose is maintaining the "prescribed" view of the current collaboration structure, i.e., creating, updating, and removing *hADL element instances* of the **hADL Runtime View** as pending to existing (i.e., *prescribed*) or pending to be released (i.e., *prescribed removed*) (see Fig. 5 right). Elements remain in the prescribed state until the corresponding change at the collaboration platform has occurred and then enter the *described* state. To this end, the CLC doesn't invoke the collaboration platforms directly but delegates any valid client request to surrogates (see below) which ultimately update the instances' status from *prescribed* to *described*.

Note that the CLC remains external to the actual ongoing collaboration. It's limited to setting up and evolving the collaboration structure. The collaboration itself, such as joint content production, chat discussions, or message authoring and dispatching, is subject to the involved users via the respective collaboration platforms. The CLC's name is inspired by software architecture terminology as it assumes the role of a *linkage connector* but at the level of collaboration entities rather than software components:

> Linkage connectors are used to tie the system components together and hold them in such a state during their operation. [...] a linkage connector may disappear from the system or remain in place to assist in the system's evolution [26, p. 168].

We introduce **Surrogates** as the key mechanism for mapping a high-level collaboration model in hADL to the implementation-level collaboration platforms. Typically, a hADL model will specify a separate surrogate for each HumanComponent, CollaborationConnector, and CollaborationObject (see Fig. 5 left). A surrogate is responsible for acquiring access to a collaborator (i.e., a ChatUser),

respectively creating an instance of a CollaborationObject (e.g., a ChatRoom), wiring up these elements, and eventually releasing them again. To this end, surrogates exhibit sophisticated capabilities around a collaboration platform's (web) API. A *DocFile* surrogate, for example, knows which GoogleDrive collaboration platform API methods to invoke in order to establish/remove a *own*, *edit*, *read*, and *comment* link with a *DocUser* as well as *templateFile* and *fileInFolder* relations. In contrast, the *DocUser* surrogate encapsulates all logic required to contact a user and invite him/her to join the collaboration structure such as becoming editor of a document, and so on. It is up to the surrogate's implementation what communication protocol to use for interacting with a collaboration platform (typically JSON/XML over HTTP) and users (typically SMTP, XMPP, or SMS). Eventually, at runtime, there exists a surrogate instance for each instance of HumanComponent, CollaborationConnector, and CollaborationObject.

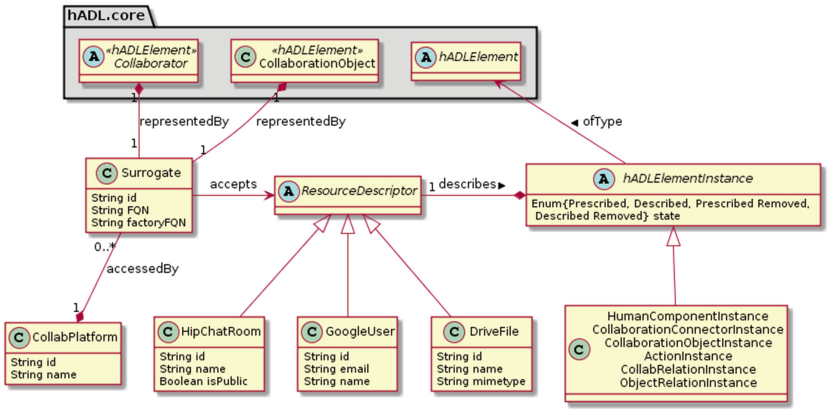

Fig. 5. Simplified UML model depicting the extensions to the core hADL model and example realizations of abstract classes. Surrogates describe what ResourceDescriptors they accept. At runtime, the CLC creates hADLElementInstances with reference to their type and their ResourceDescriptor. Subclasses of hADLElementInstance are identical to their counterpart in the hADL core model and thus are depicted as a single class for sake of brevity.

Our framework is designed to remain independent from specific process languages and engines. Hence how the process conducts the assignment of actual tasks to users is out of scope of this paper and requires process-specific mechanisms, e.g., WS-HumanTask [18] or BPMN2 user tasks [1]. In the remainder of this section, we describe how the framework's main software components interact and how to get from model to execution.

4.2 hADL Framework Component Interactions

We outline the interaction among our framework's software components based on a typical interaction sequence depicted in Fig. 6 (also found in our motivating scenario and use case implementation).

A interaction session starts with the hADL client acquiring (1) HumanComponents, CollaborationConnectors, and CollaborationObjects. Specifically, the client passes one or more tuples specifying which ResourceDescriptor describes a particular hADL element type. Here the client asks for `Bob` becoming a *ChatUser* and a *ChatRoom* with name `PreMeeting`. The hADL CLC checks the request whether hADL element types and ResourceDescriptors match (2), etc., and subsequently add instances to the hADL Runtime View (3). These instances exist in the "descriptive" state, i.e., pending to exist. The CLC then initiates the matching surrogates that will handle the individual hADL element instances (4). But first, it returns an "observable" back to the client (5).

An observable is a subscription endpoint for the client to receive events from the CLC and surrogates. We use this event-driven mechanism for asynchronous notification of successful and failed request processing. Request processing at a surrogate usually involves invoking the collaboration platform API and hence potentially requires a significant amount of time. Request completion takes even longer when the surrogate contacts a user for confirming the participation in a collaboration. A client thus doesn't block on a request but may process results (e.g., successful setup of a chat room) or react to failures (e.g., user declined to join a chat room) as these events arrive.

Next, the CLC passes all acquisition request together with the respective hADL element instance, ResourceDescriptor, and observable to the individual surrogates who process these in parallel (6,7). Surrogate A for ChatUser Bob invokes the HipChat API to check whether the user (as described in the ResourceDescriptor) already exists or has to be invited (8). In the former case the surrogate can immediately mark the hADL instance element as "descriptive", i.e., confirmed to exist (9). Subsequently, the surrogate dispatches an event back to the client (via the observable mechanism) that the acquiring was successful and includes a reference to the HumanComponentInstance representing ChatUser Bob (10). Note that the observable mechanism strongly decouples client and surrogates. The client remains unaware of surrogates—it only cares about the request outcome—and surrogates remain unaware of event consumers.

Note that from here on, we no longer depict request checks, collaboration platform invocations, or observables due to space limits.

In our example, the client continues to wireup chat room and chat user. It does so by passing the source and destination hADL instances, and link type to be established (14). Remaining at the hADL level, the CLC has no insights into how a surrogate brings about changes. Hence, for establishing links (or relations), it always triggers the surrogates of both involved endpoint instances (16,17). The surrogates' logic determines whether any action is required. For example, Surrogate B checks whether the ChatUser Bob may obtain "coordinate" capabilities and signals success (18) while Surrogate A "knows" that in

this case no action is required (the surrogate implementation assumes here that the user always agrees to become coordinator of a chat room). Note that no interaction between any two surrogates occurs for establishing links (or relations). The surrogates remain completely decoupled and any implicit information exchange occurs only via the well defined ResourceDescriptors. That is, for example, Surrogate B receives a *wire* request which contains the reference to the opposite endpoint (a HumanComponentInstance of type ChatUser, here Bob). It extracts the ResourceDescriptor—Bob's details—and thus obtains all the necessary information to determine locally (and via the collaboration platform API) whether the link may be established or not.

Note that so far no actual wiring has occurred. The client has the opportunity for further rewiring before calling *start*. Upon start (19), the CLC triggers all surrogates with pending changes to execute the rewiring (20+). Any subsequent changes require first calling *stop*. Stopping (23) signals the CLC and surrogates that the client is about to request changes to, or final releasing of, the hADL instances. A surrogate may then decide that its local view of the collaboration is outdated and pulls in the latest updates from the collaboration platform.

In our example, the client intends to release ChatUser Bob (26). The CLC marks this human component instance and all its links as "removed prescriptive" (27) and first requests all links to be removed (28,29). Unwiring works exactly like wiring. Only then does the CLC ask the surrogate to release ChatUser Bob (32). For the various CollaborationObject types, releasing typically means closing a stream, deleting or archiving a shared artifact, aborting a request, or removing a message channel. For HumanComponents and CollaborationConnectors, on the other hand, releasing implies notifying them on the ending collaboration and removing their access rights to the various collaboration object instances. Finally, upon completing the release procedure, the surrogate instance terminates (34).

4.3 From Model to Execution

From a developer's perspective, model-driven execution of collaboration structures consists of three phases: modeling the collaboration types, scripting the hADL client, and executing the collaboration structure at runtime.

The **modeling phase** comprises all activities necessary to (i) create the hADL model, (ii) specify the collaboration platform-specific ResourceDescriptors (i.e., GoogleUser, DriveFile, HipChatRoom), (iii) implement the corresponding surrogates (i.e., surrogates for DocUser, ChatUser, DocFile, Folder, and ChatRoom), and (iv) extend the hADL model with surrogate and ResourceDescriptor details (see Fig. 5). We assume in this bottom-up approach, that the utilized collaboration platforms (i.e., HipChat and GoogleDrive) already exist and expose an API suitable for invocation by the surrogates. The methodology for specifying the hADL model and aligning the surrogates is out of scope of this paper.

In the **scripting phase**, the developer implements the hADL client's logic as a set of steps that setup and modify the collaboration structure. Typically each step defines the required input (e.g., the ResourceDescriptors of the users to invite to a chat room and the chatroom's name) and the expected output,

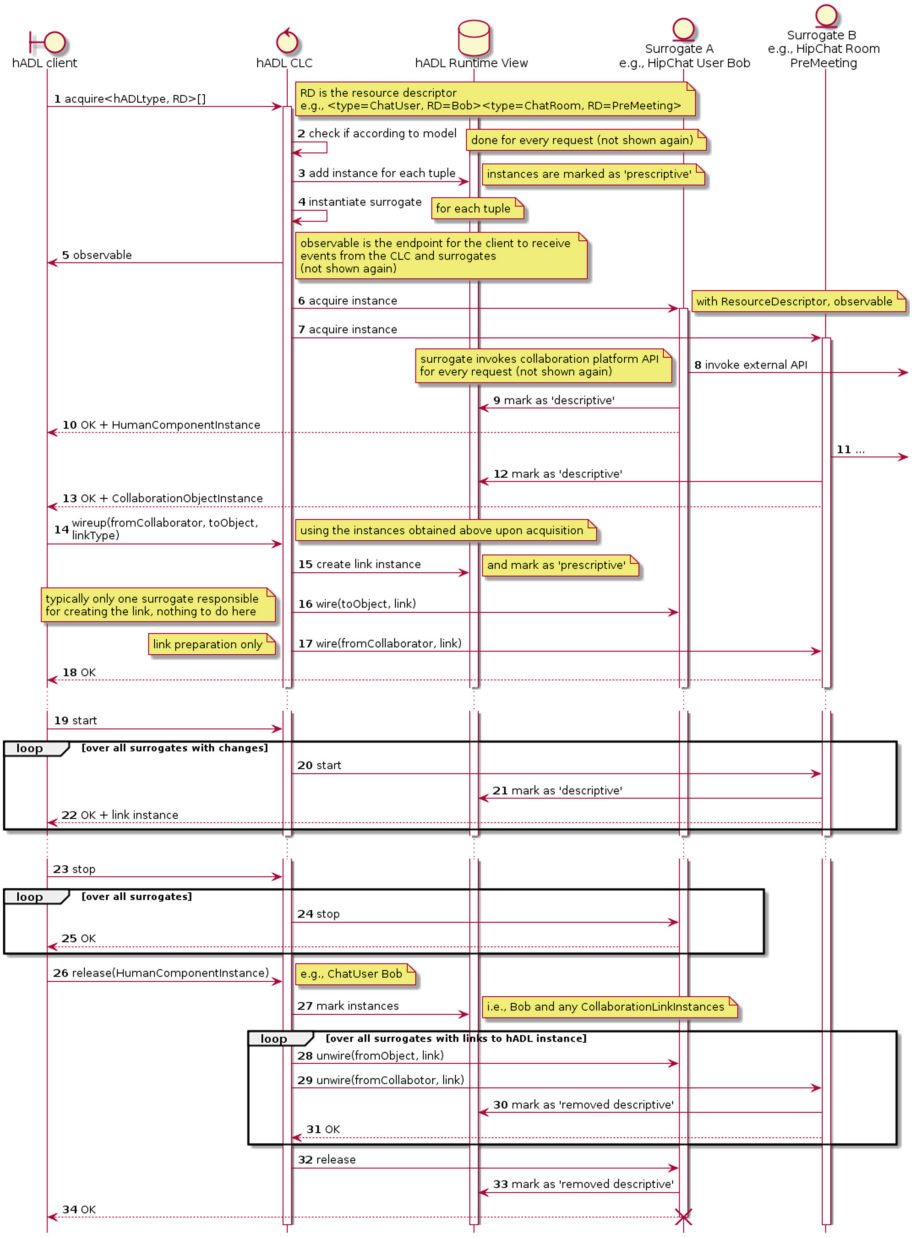

Fig. 6. Simplified sequence chart of an example interaction among hADL framework components. The hADL client requests a user and chat room, wires the user to the chat room, and ultimately removes the user again. The sequence chart is available in high resolution among the supporting material at http://wp.me/P1xPeS-6L.

i.e., the hADL element instances for use in subsequent steps (e.g., the chat user instance and chat room instance). The developer inspects the extended hADL model to learn what elements are available, how these can be linked (i.e., actions, links, and relations), and which resource descriptors match. S/he subsequently extracts the element identifiers for invoking the hADL CLC. For example, the developer learns that a GoogleUser ResourceDescriptor may be used to acquire a ChatUser and a DocUser. No insights into surrogate implementation or collaboration platform API are required. Listing 1 demonstrates how to invoke the CLC purely using model information. Note that in this listing all steps are condensed into a single script for sake of brevity. The resulting hADL client script (currently plain java) becomes integrated into the application's logic or a business process specification.

Finally, in the **execution phase** the hADL client script is executed as regular source code, requiring only that the surrogate implementations are accessible to the CLC for instantiation.

5 Use Case Implementation

We demonstrate the basic capabilities of our framework and the feasibility of our approach through the proof-of-concept implementation of a use case and hADL execution platform. Specifically, we showcase the setup, rewiring, and releasing of two distinctly different collaboration mechanisms—Google Drive documents and HipChat chat rooms—as described in the example process[1] in Fig. 1. We provide all hADL models, extensions, source code, and configurations for replicating the use case as supporting online material (SOM) available at http://wp.me/P1xPeS-6L.

We implemented surrogates for Google Drive files and HipChat chat rooms. The file surrogate makes use of the official java client for Google Drive[2], while we extended a third-party java library[3] for implementing the chat room surrogate. Both platforms automatically send notification emails to users when they obtain access to files, respectively chat rooms. Hence our HumanComponent surrogates are minimal implementations. The use case introduces ResourceDescriptors for the Google Drive file (id, name, and mime type), the HipChat chat room (id, name, and topic), and user identification (by email address, applied for Google Drive and HipChat users); see also Fig. 5 middle. Setup includes registration of the same five users for Google Drive and HipChat: two committee members, two assessment team members (for one exemplary job application), and the minority officer.

Listing 1 summarizes the hADL framework client pseudo code for supporting the process in the motivating scenario. The pseudo code lacks use of our framework's asynchronous communication mechanism (i.e., observables and events)

[1] No process engine was used for our use case implementation as this paper addresses the collaboration structure execution aspect only.

[2] https://developers.google.com/api-client-library/java/apis/drive/v2.

[3] https://github.com/evanwong/hipchat-java/tree/java7.

for sake of clarity and brevity. Note how collaboration changes are typically enforced at the begin and end of process steps: lines 1–4 describe preparations for the *Evaluation Criteria Meeting*, lines 5–11 list the post-meeting changes to document and chatroom. Lines 12–22 show the setup of the assessment team and department members with access to report and chatroom. Lines 23–28 reduce access upon the assessment deadline, and lines 29–33 setup the *Applicant Selection Meeting*, then the listing skips a few steps before lines 34–35 completely close down the collaboration instance. The full script is available in the SOM.

Listing 1. Pseudo code for managing collaboration structures in support of a hiring process: variables with 'I'-postfix are hADL model runtime instances; resource descriptors are reduced to simple strings, e.g., 'Bob'.

```
1: file1I = acquire(Model.DOCFILE,'EvalCriteriaReport') {prepare meeting}
2: users1I[] = acquire(Model.DOCUSER,['Alice','Bob']) {hiring committee}
3: link(users1I, file1I, Model.EDITING)
4: start() {ready for meeting}
5: stop() {upon meeting end}
6: unlink(users1I, file1I, Model.EDITING)
7: link(users1I, file1I, Model.COMMENTING)
8: room1I = acquire(Model.CHATROOM,'CriteriaDiscussionRoom')
9: users2I[] = acquire(Model.CHATUSER,['Alice','Bob'])
10: link(users2I, room1I, Model.CHATTING)
11: start() {chatroom setup completed}
12: stop() {assessment phase begins}
13: file2I = acquire(Model.DOCFILE,'Assessment1') {for job application 1}
14: users3I[] = acquire(Model.DOCUSER,['Carol','Dave']) {assessment team}
15: user5I = acquire(Model.DOCUSER,'Eve') {minority officer}
16: link(users3I, file1I, Model.READING) {access to eval criteria}
17: link(users3I, file2I, Model.EDITING) {access to assessment report}
18: link(users1I + user5I, file2I, Model.COMMENTING) {commenting access for department members and
    minority officer}
19: room2I = acquire(Model.CHATROOM,'Application1DiscussionRoom') {application specific discussion
    room}
20: users4I[] = acquire(Model.CHATUSER,['Carol','Dave']) {acquire remaining department members}
21: link(users2I + users4I, room2I, Model.CHATTING) {all department member may discuss}
22: start() {assessment scope setup completed}
23: stop() {assessment deadline reached}
24: unlink(users3I, file2I, Model.EDITING)
25: unlink(users1I, file2I, Model.COMMENTING)
26: link(user1I+users3I, file2I, Model.READING) {read access for department members, commenting remains
    for officer}
27: release(room2I) {close chatroom for application 1}
28: start() {execute changes}
29: stop() {before selection meeting}
30: file2I = acquire(Model.DOCFILE,'CandidateList')
31: link(users1I, file3I, Model.EDITING)
32: link(user5I, file3I, Model.COMMENTING) {minority officer can comment before meeting completion}
33: start() {execute changes}
34: stop() {skipping steps here ...}
35: releaseAll() {... ultimately, shutting down collaboration: files and chatroom}
```

6 Related Work

Managing human work dependencies is not limited to processes. Brambilla and Mauri integrate social network-centric actions into web applications via social primitives [5]. Their focus is on making commenting, posting, voting, and searching capabilities of public social platforms available as WebML operations. Our approach, in contrast, focuses on specifying and executing the collaboration structures, leaving the actual collaboration per se to the users via the actual, underlying platforms. Activity-centric approaches such as [2,12] put control into the hands of users for flexibly defining and deviating from (ad-hoc) processes.

Human and Artifact-centric BPM. Even traditional workflow description languages dedicated to modeling human involvement such as Little-JIL [6], BPEL4People [15], or WS-HumanTask [18] foresee no explicit communication among process participants outside of tasks. Although BPEL4people supports four eyes, nomination, escalation, and chained execution scenarios—and WS-HumanTask allows attaching comments to tasks—all interaction is purely task-centric. Similarly, La Rosa et al. [20] demonstrate how EPC-based models may involve multiple users in a task including artifacts but neither how multiple participants collaborate, nor their capabilities on the artifacts. In contrast, Liptchinsky et al. model the impact of social relations on software artifacts and the respective engineering process [21]. The collaboration mechanisms that give rise to social relations and process execution support remain out of scope. Subject-oriented BPM [14] models all data flow exclusively with messages between process participants. Hence other collaboration mechanisms such as shared artifacts, chat rooms, etc. are extremely awkward to represent.

Artifact-centric BPM approaches [17] (aka document-centric, data-centric, or object-centric) focus on specifying artifact structure, states, and access rights. Examples such as the Business Entity Definition Language [23], Philharmonicflows [19], FlexConnect [24] or ad-hoc processes driven by documents [8] remain restricted to artifacts and leave aside other collaboration mechanisms such as chatting, voting, or direct messaging. These approaches, however, model artifacts in much more detail compared to hADL.

Social BPM and Crowd Sourcing. Recent research efforts started explicitly targeting the integration of social media into business process management (BPM) technology. Brambilla et al. present design patterns for integrating social network features in BPMN [4]. A social network user may engage in task-centric actions such as voting, commenting, reading a message, or joining a task. Böhringer utilizes tagging, activity streams, and micro-blogging for merging ad-hoc activities into case management [3]. Dengler et al. utilize Wikis and social networks for coordinating process activities [7]. oBPM [16] relies on task and artifact abstractions for coordinating business process modelling.

These approaches differ in several crucial aspects from our work: (i) they integrate collaboration mechanisms only in single tasks, (ii) these mechanisms are typically hard-wired social media connectors with no abstraction, (iii) and/or collaboration aspects support the process design phase [13] only.

To the best of our knowledge, no contemporary research approaches address the issue of modeling and executing collaboration structures. We focused in our own, previous work on establishing a passive runtime view of the ongoing collaboration from monitoring a system's software architecture [10] and addressed the aspect of configuration and deployment of collaboration systems, i.e., provisioning the technical infrastructure [25]. Our approach in this paper is completely independent of either works. The discussed work above presents primarily orthogonal approaches worthwhile investigating for future integration.

7 Conclusions and Outlook

In this paper[4], we presented a first framework for model-driven execution of collaboration structures. We demonstrated how to specify collaboration structures on an abstract level subsequently grounded in concrete collaboration platforms via surrogates. The preliminary evaluation use case demonstrated the application of our framework for supporting a hiring process via Google Drive documents and HipChat chat rooms. The current implementation puts a significant burden on the framework client for error handling and correct model usage. Future work will explores the use of a Domain-Specific Language for expressing and generating the source code for type safe collaboration modification. Additionally, we will focus on adding sophisticated error handling strategies and investigating the integration with a process engine.

References

1. BPMN 2.0. http://www.omg.org/spec/BPMN/2.0/PDF/
2. Bernstein, A.: How can cooperative work tools support dynamic group process? bridging the specificity frontier. In: Proceedings of ACM Conference on Computer Supported Cooperative Work, CSCW 2000, pp. 279–288. ACM, New York (2000)
3. Böhringer, M.: Emergent case management for ad-hoc processes: a solution based on microblogging and activity streams. In: zur Muehlen and Su [22], pp. 384–395
4. Brambilla, M., Fraternali, P., Vaca, C.: BPMN and design patterns for engineering social BPM solutions. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 219–230. Springer, Heidelberg (2012)
5. Brambilla, M., Mauri, A.: Model-driven development of social network enabled applications with WebML and social primitives. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 41–55. Springer, Heidelberg (2012)
6. Cass, A.G., Lerner, B.S., Sutton Jr., S.M., McCall, E.K., Wise, A.E., Osterweil, L.J.: LittleJIL/Juliette: a process definition language and interpreter. In: Ghezzi, C., Jazayeri, M., Wolf, A.L. (eds.) ICSE, pp. 754–757. ACM (2000)
7. Dengler, F., Koschmider, A., Oberweis, A., Zhang, H.: Social software for coordination of collaborative process activities. In: zur Muehlen and Su [22], pp. 396–407
8. Dorn, C., Dustdar, S.: Supporting dynamic, people-driven processes through self-learning of message flows. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 657–671. Springer, Heidelberg (2011)
9. Dorn, C., Taylor, R.N.: Architecture-driven modeling of adaptive collaboration structures in large-scale social web applications. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 143–156. Springer, Heidelberg (2012)
10. Dorn, C., Taylor, R.N.: Coupling software architecture and human architecture for collaboration-aware system adaptation. In: Notkin, D., Cheng, B.H.C., Pohl, K. (eds.) ICSE, pp. 53–62. IEEE/ACM (2013)
11. Dorn, C., Taylor, R.N.: Analyzing runtime adaptability of collaboration patterns. Concurrency Comput. Pract. Experience **27**(11), 2725–2750 (2015)

[4] This research was partially supported by the EU FP7 SmartSociety project (600854).

12. Dustdar, S.: Caramba process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. Distrib. Parallel Databases **15**(1), 45–66 (2004)
13. Erol, S., Granitzer, M., Happ, S., Jantunen, S., Jennings, B., Johannesson, P., Koschmider, A., Nurcan, S., Rossi, D., Schmidt, R.: Combining BPM and social software: contradiction or chance? J. Softw. Maint. Evol. Res. Pract. **22**(6–7), 449–476 (2010)
14. Fleischmann, A., Schmidt, W., Stary, C., Obermeier, S., Börger, E.: Subject-Oriented Business Process Management. Springer, Heidelberg (2012)
15. Ford, M., Endpoints, A., Keller, C.: WS-BPEL extension for people (BPEL4People), version 1.0 (2007)
16. Grünert, D., Brucker-Kley, E., Keller, T.: oBPM an opportunistic approach to business process modeling and execution. In: Workshop on Business Process Management and Social Software (BPMS2 2014) (2014)
17. Hull, R.: Artifact-centric business process models: brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
18. Ings, D., Clement, L., König, D., Mehta, V., Mueller, R., Rangaswamy, R., Rowley, M., Trickovic, I.: Web services human task (WS-HumanTask) specification version 1.1. Technical report, OASIS, July 2012. http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.html
19. Künzle, V., Reichert, M.: Philharmonicflows: towards a framework for object-aware process management. J. Softw. Maint. Evol. Res. Pract. **23**(4), 205–244 (2011)
20. La Rosa, M., Dumas, M., ter Hofstede, A.H.M., Mendling, J., Gottschalk, F.: Beyond control-flow: extending business process configuration to roles and objects. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 199–215. Springer, Heidelberg (2008)
21. Liptchinsky, V., Khazankin, R., Truong, H.-L., Dustdar, S.: A novel approach to modeling context-aware and social collaboration processes. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 565–580. Springer, Heidelberg (2012)
22. zur Muehlen, M., Su, J. (eds.): BPM Workshops. LNBIP, vol. 66. Springer, Heidelberg (2011)
23. Nandi, P., Koenig, D., Moser, S., Hull, R., Klicnik, V., Claussen, S., Kloppman, M., Vergo, J.: Data4BPM, part 1: introducing business entities and the business entity definition language (BEDL), April 2010. http://ibm.co/1QTR0IH
24. Redding, G., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: A flexible, object-centric approach for business process modelling. SOCA **4**(3), 191–201 (2010)
25. Sungur, C.T., Dorn, C., Dustdar, S., Leymann, F.: Transforming collaboration structures into deployable informal processes. In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (eds.) ICWE 2015. LNCS, vol. 9114, pp. 231–250. Springer, Heidelberg (2015)
26. Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture: Foundations, Theory, and Practice. Wiley Publishing, New York (2009)