Architecture-Driven Modeling of Adaptive Collaboration Structures in Large-Scale Social Web Applications

Christoph Dorn and Richard N. Taylor

Institute for Software Research, University of California, Irvine, CA 92697-3455
[cdorn|taylor]@uci.edu

Abstract. Internet-based, large-scale systems provide the technical foundation for massive online collaboration forms such as social networks, crowdsourcing, content sharing, or source code generation. Such systems are typically designed to adapt at the software level to achieve availability and scalability. They, however, remain mostly unaware of the changing requirements of the various ongoing collaborations. As a consequence, cooperative efforts cannot grow and evolve as easily nor efficiently as they need to. An adaptation mechanism needs to become aware of a collaboration's structure and flexibility to consider changing collaboration requirements during system reconfiguration. To this end, this paper presents the human Architecture Description Language (hADL) for describing the envisioned collaboration dynamics. Inspired by software architecture concepts, hADL introduces human components and collaboration connectors for describing the underlying human coordination dependencies. We further outline a methodology for designing collaboration patterns based on a set of fundamental principles that facilitate runtime adaptation. An exemplary model transformation demonstrates hADL's feasibility. It produces the group permission configuration for MediaWiki in reaction to changing collaboration conditions.

Keywords: Design Tools and Techniques, Collaboration Patterns, Adaptation Flexibility.

1 Introduction

The last two decades have witnessed the emergence of numerous web-based, large-scale collaboration tools. Web sites appeared for diverse purposes such as social networking (e.g., Facebook, LinkedIn), collaborative tagging (e.g., Digg), content sharing (e.g., YouTube, Flickr), knowledge creation (e.g., Wikipedia), crowdsourcing (e.g., Amazon Mechanical Turk), or source code production (e.g., GitHub, SourceForge).

Users of such social Web applications typically face one major problem: a rigid, limited set of available collaboration mechanisms in a one-size-fits-all manner. Interaction means such as direct messaging, group chats, discussion boards, task assignments, or shared artifacts remain independent of the collaboration's scale and complexity and thus form a constraint on how large a joint effort can grow, how easily and how efficiently it may evolve. Amazon MTurk, for example, scales the Master/Worker pattern to thousands of users and tasks. The MTurk platform implements a rigid interaction pattern where communication amongst participants is not foreseen. Hence, pattern adaptation for supporting more complex collaborations that require coordination between individual workers is impossible.

We claim that an explicit model of collaboration structures is of uttermost importance for describing a collaboration system's flexibility and subsequently supporting the evolution of collaborative efforts through pattern adaptation.

We take inspiration from software architectures to address this problem for large-scale collaboration systems. A system's software architecture as described in terms of components and connectors has a profound effect on its adaptability, especially scalability [19]. The same holds true for human collaboration (see Sec. 4). Connectors in the form of humans (e.g., forum moderators, secretaries) and software services (e.g., mailing lists, task lists) manage dependencies between collaborators (i.e., human components) when direct interaction amongst all participants is no longer viable. The explicit modeling of humans as components and connectors—a distinction which existing approaches have insufficiently addressed so far (see Sec.3)—draws the focus to the collaboration structure's flexibility and thus facilitates adaptation.

Our contribution in this paper is three-fold. We (i) introduce the human Architecture Description Language (hADL) in Sec. 5, (ii) provide a methodology for defining adaptable collaboration patterns in Sec. 6, and (iii) demonstrate the model's feasibility based on an exemplary model-to-configuration transformation in Sec. 7. We find that not only are components and connectors a very suitable abstraction mechanism for describing collaboration patterns and their adaptation flexibility. As integral part of a human architecture, they also successfully support pattern evolution.

2 Motivating Scenario

Suppose a research project integrates knowledge from the wider research community in the form of Wiki-style articles. After the infrastructure for collecting and managing user contribution goes online, participation remains low but stable. One regular project staff member quality checks changes to existing articles and browses through the content list to check new article entries.

Soon, a report in the media about the research project sparks wide-spread interest with subsequent participation levels soaring. This has significant implications on the quality assurance procedure which has to deal with vandalized or spammed articles. Conflicting opinions amongst contributors of the same article lead to editing wars. A single quality manager is no longer up to the task. Simple replication of her role is one option, but changing the collaboration pattern is potentially more effective. Multiple options exist to handle articles exhibiting high revision rates: (i) updates are checked by an expert — possibly crowdsourced — to decide upon article rollbacks, (ii) contributors vote on changes, or (iii) experts discuss and negotiate changes. Alternatively, articles subject to update wars are temporarily protected or receive a limited write quota. New articles still need no approval to keep participation barriers low but observers now receive notifications about new entries. Depending on the rate of new articles, such monitoring itself may require topic-based subscriptions to ensure that observers receive only notifications relevant to their interests. Planning and subsequently implementing such restructuring requires an explicit model of the underlying collaboration structure and its adaptation flexibility.

3 Related Work

Research efforts that specifically focus on social or collaborative aspects in large-scale systems are still rare. Existing research addresses mainly the general idiosyncracies of Web 2.0 but remains unaware of specific interaction structures at runtime [21]. Model-driven Web engineering approaches so far focus primarily on software aspects [16] and don't go beyond (user) context-centric adaptations [1]. Requirements elicitation and specification approaches consider collaboration (e.g., CSRML [20]) or adaptation (e.g., [17]) but omit the effects of patterns on adaptation flexibility.

Activity-centric frameworks (e.g., [6,12]) define tasks and their relations for integrating humans and software components [2]. Human-centric workflow systems define business artifacts, their transformations, and interdependencies [8]. The Business Entity Definition Language [13], for example, aggregates access rights, data structure, object state transitions, and events. The human collaboration structure, however, remains implicit. Similar, the Business Process Modeling Language (BPMN [22]) describes human tasks and their dependencies. Recent research efforts on large-scale workflow deployment such as Human-provided Services (HpS) [15], Turkit [9], or CrowdLang [11] differ in their degree of formalizing complex workflows that go beyond simple task assignment in Amazon Mechanical Turk.

Most of these models, tools, and frameworks contain (model) elements similar to hADL (e.g., when specifying human roles and their associated capabilities) but differ in two crucial aspects. First, all these approaches lack an explicit distinction between human components and collaboration connectors. Consequenlty any adaptation knowhow tailored either to coordination or to work execution remains implicit and hidden within each platform. Second, most collaboration platforms focus on a particular collaboration pattern and the associated limited set of adaptation capabilities. Process-centric models, for example, focus only on task execution, no matter whether ad-hoc or rigidly specified. They cannot be applied for describing other patterns such as co-authoring Wiki articles or spreading news on twitter and vice versa.

Extensible software architecture description languages (ADLs e.g., [7,3]) emerged from the need to rigorously define the language's semantics while remaining flexible enough to address the specific needs of a particular domain. Augmenting an existing ADL to describe all details of the human collaboration patterns, however, would be cumbersome as software structure and human interactions reside on different conceptual levels. Nevertheless, ADLs provide fundamental principles that inspire and guide our human architecture description language hADL.

4 The Case for a Human Architecture

The observation that software systems and human collaborations share the same challenges in managing dependencies inspired our concept of a human architecture. Both domains require coordination of (i) shared resources, (ii) producer/consumer relationships, (iii) simultaneity constraints, and (iv) task/subtask relations [10]. An architecture describes how a system addresses these challenges. In the domain of software engineering, following definition of a software architecture fits equally well to collaborative efforts: "A software system's *architecture* is the set of principal design decisions made about the system.“ [18], p.58.

Components and connectors are the primary building blocks of a software architecture. Components are the loci of computation and data management whereas connectors facilitate and control the interactions between components. Roles such as managers, team leaders, secretaries are rarely described as connectors but they perform a similar task: the coordination of other human (i.e., human components). Highlighting further similarities: in software architecture, architectural styles consist of a set of development context dependent design decisions, constraints, and resulting properties. Collaboration patterns correspondingly describe what combination of human components and coordinators are suitable for a particular joint effort [5,4].

In software architectures, connectors are the key element to system adaptability. For example, connectors allow the dynamic replacement of behavior components in robotic systems without affecting other components. Web proxies are connectors on the Internet that decide which server (component) should process a particular client (component) request. Overloaded or unavailable servers thus become transparent to the client. In the scenario, the article contributors and readers constitute the human components. (Human) quality managers and (software) change monitors implement connector functionality for managing the read and write dependencies amongst the human components. The importance of collaboration connectors grows with the scale and complexity of joint efforts especially in distributed settings where individual collaborators have little opportunity for informal communication.

5 The Human Architecture Description Language

The core human Architecture Description Language (hADL) defines collaborators, their means of interaction through messages, streams, and shared artifacts, and dependencies amongst collaboration objects (Fig. 1). We explain the individual elements based on a hADL model instance (Fig. 2) for the motivating scenario.

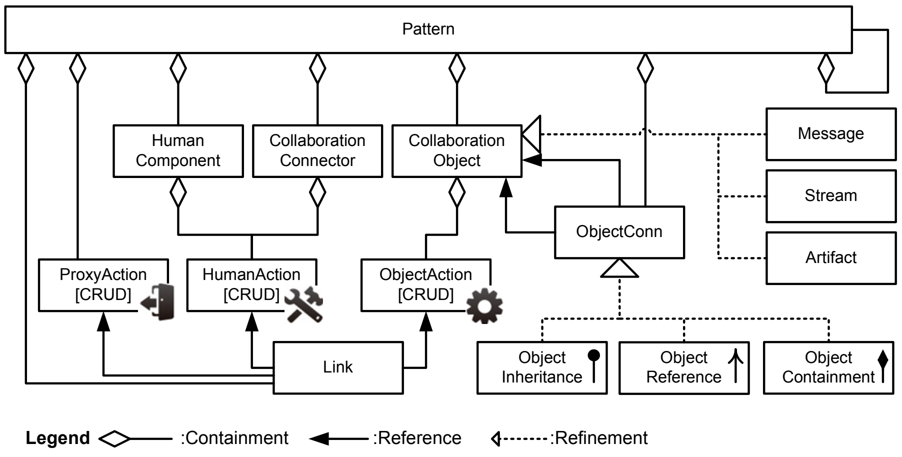

Fig. 1. hADL model (symbols in ObjectConn subtypes and Actions represent the respective visualization in model instances.)

A human architecture describes the configuration of *HumanComponents* and *CollaborationConnectors* to fulfill a particular purpose, for example: carrying out a task, creating a shared artifact, or negotiating a leader. The architecture's purpose determines a suitable collaboration *Pattern.* Typical patterns include Master/Worker, Publish/Subscribe, Shared Artifact, and Peer-to-Peer (e.g., [4]). A HumanComponent has a particular collaboration role that is essential to the completion of the collaborative effort (e.g., Contributor, Reader, Observer in Fig. 2 left and right). A CollaborationConnector provides coordination capabilities to HumanComponents within the pattern's scope (e.g., QualityManager, VandalismDetector, ArticleMonitor in Fig. 2 center). A CollaborationConnector covers the full spectrum from purely human, to software-assisted, to purely software implemented. In the scenario, a quality manager manually approving all edits illustrates a human collaboration connector. In contrast an article monitor notifying users via email about updates exemplifies a software-based collaboration connector.

HumanComponents and CollaborationConnectors are the active collaboration elements in hADL, but they don't specify the means of collaboration. When physically distributed, humans usually communicate through *Messages*, *Streams*, or shared *Artifacts.* The hADL model considers these three types as *CollaborationObject* variants. A Message is a onetime, immutable object exchanged between a set of collaborators (components and connectors), a typical example is an email. A Stream is a series of messages where sender and receiver maintain a temporary relationship. Two broad types exist: (a) subscriptions characterize a set of independent messages (such as news items in RSS feeds or updates on a user's facebook wall). Alternatively, (b) multimedia streams consist of dependent messages (i.e., frames) that constantly refresh the receiving end (e.g., video chat). A (shared) Artifact is a long-living object that is subject to (si-

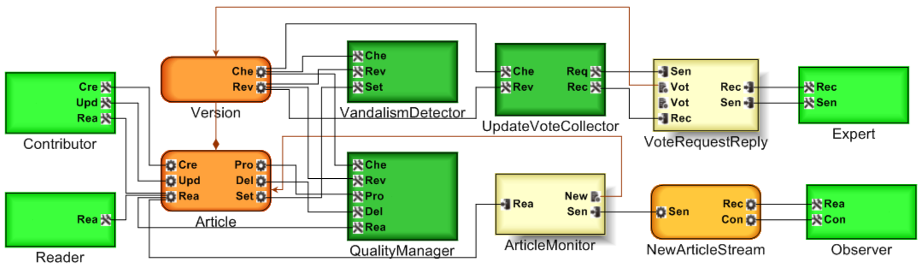

Fig. 2. Scenario hADL model instance: components as light-green shaded boxes, connectors as dark-green shaded boxes, collaboration objects with rounded corners, and substructure patterns with shadow (colors online). Icons represent human, respectively object actions.

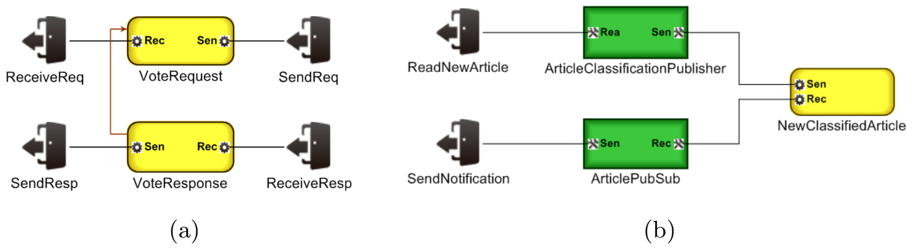

Fig. 3. hADL models for (a) Vote Request Reply substructure and (b) Topic-based Article Monitoring substructure.

multaneous) manipulation by multiple collaborators. In the scenario, respective examples are (i) emails sent to Experts to vote on article updates (Fig. 3a), (ii) notifications about new articles (Fig. 3b), and (iii) the articles themselves (Fig. 2). *ObjectConns* describe dependencies amongst CollaborationObjects such as refinement (*ObjectInheritance*), relation (*ObjectReference*), and substructure (*ObjectContainment*). Note that ObjectConns merely highlight such dependencies to improve pattern comprehension but they don't replace data modeling.

The choice of communication means has a profound impact on the collaboration and thus needs to be made explicit. Hence, hADL requires a CollaborationObject between any two or more HumanComponents and/or CollaborationConnectors. This is in contrast to traditional ADLs (e.g., xADL [3] or ACME [7]) where component interfaces link directly to connector interfaces. A rough software architecture interface equivalent in hADL is the *Action*. HumanComponents and CollaborationConnectors exhibit *HumanActions* that specify what access rights a collaborator requires to fulfill its role, whereas a CollaborationObject has *ObjectActions* for defining what rights it grants to particular collaborator. An Action distinguishes between *Create*, *Read*, *Update*, and *Delete* (CRUD) privileges. The article Contributor in Fig. 2, for example, exhibits an *Edit* action with Create, Update, and Read rights. Ultimately, CRUD rights need to match when a *Link* connects a HumanAction with an ObjectAction. Multiple Collaborators may connect to the same ObjectAction when they share the same manipulation rights (e.g., several CollaborationConnectors in Fig. 2 connect to the same Article *Read* action).

In some cases, we wish to introduce substructures to hide low-level collaboration details that are irrelevant at the higher-level collaboration scope. In the scenario, a CollaborationConnector monitors new Articles. Whether this connector merely sends an email to all interested Observers or whether observers subscribe to certain article topics is described at a lower level. In the latter case, the substructure defines the appropriate subscription mechanism (Fig. 3b). Pattern substructures are equally well suited to hide complex CollaborationObjects (e.g., tightly coupled request and response messages for voting on article changes, Fig. 3a). In hADL, such substructures are implemented as recursive embedding of Patterns with the use of *ProxyActions*.

6 Designing for Adaptation

Research in software architectures supplies several concepts and tools for designing and analyzing collaboration structures. In our previous work ([5,4]), we applied the BASE framework [19] for studying the adaptation flexibility of various collaboration patterns. Based upon the insights gained in our recent analysis and our experience in architecture-based software adaptation we propose a set of principles that facilitate collaboration adaptation. Specifically, these principles build in part upon an earlier discussion of dynamic software adaptability in the scope of architectural styles [14] and provide best practices when using hADL.

Identifying Adaptable Elements: Collaborative behavior can be modeled at multiple levels of abstraction: from an organization, a department, a team, an individual human, down to a single user's behavior strategies. The finest abstraction level determines the lowest possible level of adaptation. In the presence of modeled, identifiable user behavior, we are able to execute adaptations in the form of recommendations. For example we may suggest switching from "locking an artifact for editing it" to "issuing small but frequent article updates without locking". In contrast, we cannot reconfigure a non-performing team internally but we have to replace it as a whole when the most detailed level merely describes teams. In hADL, pattern substructures allow the simultaneous modeling of multiple abstractions level.

Encapsulating Elements: Collaboration adaptability greatly increases when elements (components, connectors, objects) are easy to replace. Encapsulation describes how tightly an element is woven into its surrounding environment. A worker in the Master/Worker pattern only knows about his personal task copy and about the assignment connector he obtained the task from. This makes him easily replaceable as the assignment connector merely needs to provide a task copy to another worker. In contrast, a group of authors that exchange article drafts directly via email exhibits tight coupling. Removing one author requires considerable effort: notification of all other authors, synchronizing of progress, and ensuring orderly handover of unfinished tasks to the remaining co-authors.

A suitable collaboration pattern in this situation may encourage encapsulation through various mechanisms. For example, replacing direct messages with a shared artifact relieves an individual author from keeping track of involved contributors. Introducing a collaboration connector for continuous integration of individual article sections further limits the coordination dependencies amongst authors. Clearly identified and assigned roles (lead author, data collection, proof reading, figure design, etc) within the group additionally promotes encapsulation. hADL avoids enforcing a particular collaboration pattern. Instead, hADL enables the system designer to flexibly assemble a suitable composition of components, connectors, and collaboration objects.

Just as software architectures suffer from implementations that don't follow the prescribed architectural style at code level, so are informal communication channels jeopardizing the adaptation characteristics of a collaboration pattern. The most adaptive pattern will exhibit potentially catastrophic adaptation consequences when the involved users circumvent the foreseen communication and coordination means and fall back onto multipurpose, pattern external communication channels such as email. The underlying collaboration infrastructure needs discouraging the use of external channels. Strategies are pattern specific, for example, hiding other collaborators, anonymizing collaborators, or providing incentives to communicate within the system.

Controlling Interaction: Fostering encapsulation is one principle that simplifies element replacement. Controlling an element's interactions with its environment is equally important. Coordination dependencies become clear and thus manageable when collaborators utilize explicit interactions. Take as an example a worker producing the input for another worker: transferring the output via precisely specified messages clearly identifies the involved actor roles. Collaboration interdependencies, however, remain largely hidden when such interactions occur via a shared artifact. Connectors are able to provide dedicated support for each interaction type only in the former case. hADL promotes the use of connectors where sensible but does not require them when deemed unnecessary.

Managing State: When replacing a human, we need to address what needs to happen with that user's internal collaboration state. An assignment connector might be waiting for task responses or has unassigned task requests still in his inbox. An article author might be currently working on an unfinished section. Three basic strategies address this challenge: (i) ignore existing state (i.e., work progress) and provide some form of compensation, (ii) provide mechanisms that facilitate the externalization of collaboration state such as shared artifacts or dedicated work progress messages, and (iii) split activities into such fine-grained parts that adaptation may be postponed until completion. hADL encourages the use of collaboration objects to render state explicit but currently lacks support for modeling component or connector internal state.

Making Bindings Malleable: Late binding in collaboration patterns delays addressing of messages until their destination absolutely needs to be determined. In a workflow, for example, the worker carrying out a particular task remains undetermined until shortly before task assignment. In the scenario, experts become part of a voting group just shortly before they are actually needed for deciding on an article update. Shared artifacts yield similar decoupling as contributors need not be known in advance. Patterns with such built-in flexibility allow for adaptation decision just in time. Collaboration objects in hADL constitute specification points for implementation specific addressing mechanisms, thus facilitating just-in-time bindings.

7 Evaluation

In this section, we show that hADL is suitable for capturing flexible collaboration patterns by modeling the MediaWiki platform[1]. Subsequently, we demonstrate runtime dynamic reconfiguration of MediaWiki's underlying collaboration pattern. To this end, we briefly present modeling tool support and then provide the MediaWiki hADL model including its mapping onto explicit and implicit group permissions. The hADL model, introduced model instances, and transformations are available for download at `http://wp.me/P1xPeS-2h`.

We adopted the Generic Modeling Environment[2] (GME) for designing, visualizing, and manipulating the hADL model and model instances. GME provides an automatic model update mechanism that allows for rapid, iterative refinement of the hADL model and model instances. The hADL model, therefore, provides only core elements for describing human collaboration architectures. We outline below how extensions cover domain-specific requirements that are otherwise insufficiently addressed. For most changes of the hADL model, the GME model update mechanism is able to successfully upgrade existing model instances to take advantage of problem-specific extensions.

7.1 Modeling MediaWiki

MediaWiki is the underlying technology platform for Wikipedia (and many other Wikis). Figure 4 visualizes how the project wiki from the scenario might initially be set up. The collaboration objects (Page, TalkPage, WikiPage, ImageOrFile, and Revision) remain the same for all MediaWiki installations as they represent the core MediaWiki collaboration capabilities. The MediaWiki group permissions[3] are a good starting point to define the various actions the collaboration objects make available to human components and collaboration connectors. The permissions, however, are insufficient to grasp the complete collaboration pattern as they include only explicitly defined user rights. Any logged-in user, for example, has access to her WatchList but no corresponding permission exists. We, therefore, add actions (i.e., implicit permissions) that model the streaming of article changes to *ArticleObservers* via the watch list (*WatchListStream*) or notification emails (*NfyEmailStream*). Applying the design methodology from Section 6, we analyze the adaptation flexibility of MediaWiki in general and of this specific instance in particular.

[1] `http://www.mediawiki.org/wiki/MediaWiki`
[2] `http://w3.isis.vanderbilt.edu/Projects/gme/`
[3] `http://www.mediawiki.org/wiki/Manual:User_rights_management`

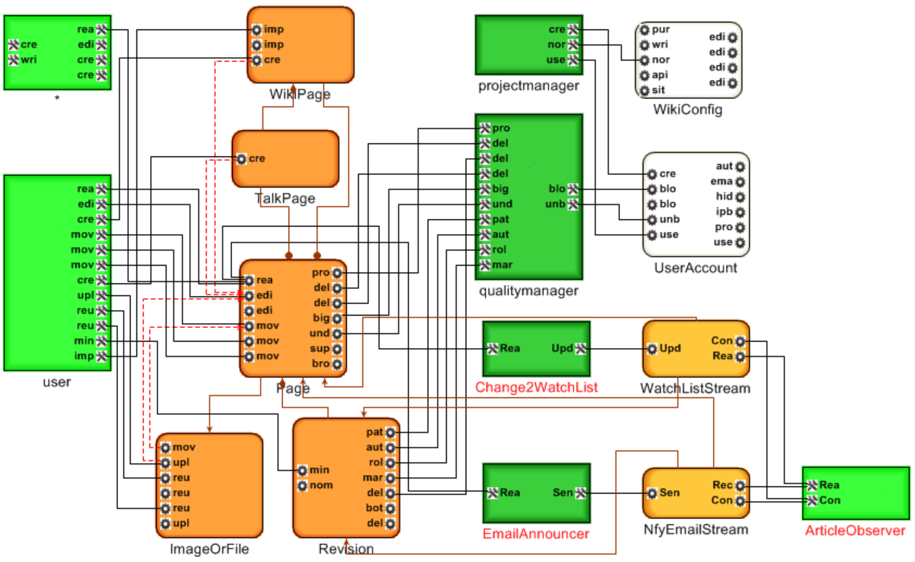

Fig. 4. MediaWiki hADL model for the initial scenario structure

Identifying Adaptable Elements. The smallest, adaptable elements in a MediaWiki installation are individual user and pages (i.e., articles). Structural adaptation actions consist of restructuring user types and (re)assigning users to particular types (i.e., groups). We won't discuss more fine-grained, build-in actions such as blocking a user or protecting a page.

Encapsulating Elements. The individual Wiki authors (component *user*) and readers (component '$\star$') exhibit strong encapsulation as all interactions happen via Wiki pages. Discussions on content, structure, etc. are equally restricted to editing of a shared artifact: the respective article TalkPage. *ArticleObservers* receive change notifications without having to rely on authors signaling updates.

Controlling Interaction. For the purpose of writing articles, MediaWiki provides sufficiently precise (inter)actions. Our scenario configuration clearly separates the various components and connectors: authors have edit, move, and upload permissions while *quality managers* have patrol, rollback, revert, delete and protection permissions. There is little to no permission overlap.

Managing State Collaboration state becomes externalized in the form of the Wiki page. A Wiki encourages publishing of frequent and small updates which enables rapid changes in author involvement.

Making Bindings Malleable. Quality managers check (i.e., patrol) article changes by inexperienced and new authors. Which particular quality manager will approve or revert a change, however, is a-priori unknown.

These characteristics and the distinction of human components from collaboration connectors facilitates reconfiguration actions to have minimal effect on active human components. As we will demonstrate next: readers, observers, and authors maintain (largely) the same rights despite considerable pattern evolution.

7.2 Dynamic Structural Adaptation

The scenario highlighted how adding, removing, or replacing users becomes insufficient to address fundamental environmental changes. Figure 5 depicts the evolved MediaWiki structure addressing the needs of the later scenario phase. The adapted structure exhibits new human components and new, reconfigured, or replaced collaboration connectors. Specifically, previous users become *experts*, new users obtain only a limited permission set. The quality managers transfer user blocking privileges to *moderators* and a software-based *editvotecollector* (collaboration connector) contacts *article guardians* for voting on user edits. Instead of receiving emails for all new articles, observers are able to configure topics of interest: the *TopicEmailAnnouncer* replaces the *EmailAnnouncer*.

Planning for reconfigurations is one benefit of modeling MediaWiki with hADL. Another potential use is describing where and how bots as well as extensions provide new functionality. Such additional components and connectors (e.g., TopicEmailAnnouncer) may build upon different collaboration patterns. Here, hADL facilitates the analysis of adaptation implications.

In the case of MediaWiki, hADL goes beyond merely describing the collaboration structure. We developed a model transformation for demonstration purposes that takes the hADL model and generates the group permissions configuration for MediaWiki. Specifically, we export the hADL model as an XML file and

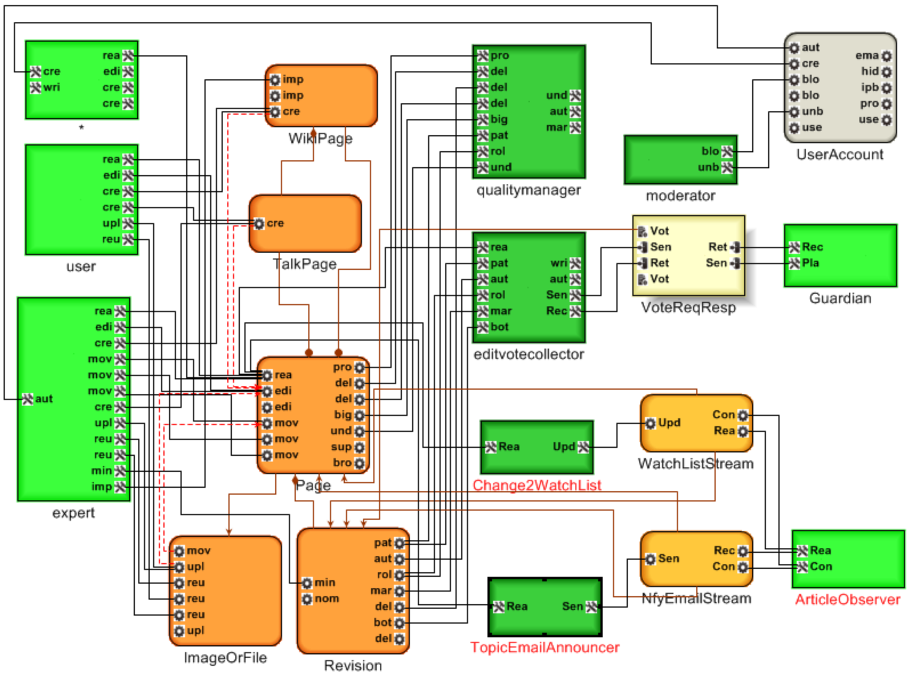

Fig. 5. MediaWiki hADL model for the evolved scenario structure

then process it with the Java Emitter Templates (JET) framework[4]. The transformation interprets every component and connector as a permission group. Each HumanAction becomes an allowed permission when connected to the corresponding ObjectAction, otherwise the permission is denied. Listing 1.1 provides the group permissions for the anonymous user group ('$\star$') in Figure 5. The resulting configuration should not include implicit rights and neither components or connectors that require no *groupPermission* representation (e.g., ArticleObservers, Change2WatchList). To this end, we extend the hADL model with additional properties. The transformation mechanism will thus ignore actions with *isImplicitRight=true* and components and connectors with *isWikiGroup=false*. We also introduce a *Requires* connection in the hADL model (dashed, red lines in Fig. 4 and Fig. 5) for highlighting dependencies between user permissions (e.g., by linking the *move* action to the *edit* action.)

```
$wgGroupPermissions['*']['createaccount']    = false;
$wgGroupPermissions['*']['read']             = true;
$wgGroupPermissions['*']['edit']             = false;
$wgGroupPermissions['*']['createpage']       = false;
$wgGroupPermissions['*']['createtalk']       = false;
$wgGroupPermissions['*']['writeapi']         = false; ...
```

Listing 1.1. GroupPermissions for anonymous MediaWiki users, i.e., '$\star$'

7.3 Discussion

Currently hADL has two main limitations. First, it lacks platform specific models. The evaluation above demonstrates hADL's feasibility but we cannot claim a general purpose tools set for various web platforms. Second, hADL features no integration with existing web modeling methodologies yet. This shortcoming, however, highlights hADL's biggest potential: a recent survey of web modeling approaches emphasizes insufficient support for sophisticated behavioral modeling [16]. Here, hADL would fit in alongside use cases, activity diagrams, or sequence diagrams to enhance current approaches such as WebML, Hera, UWE, or OOWS [16].

Even without such integration, hADL offers considerable benefits at the current stage. An explicit human architecture introduces a collaboration perspective and thus gives stake-holders another means for communicating requirements during the design process. This also enforces a structured approach to explicitly defining adaptation capabilities at the collaboration level. Being implementation independent, hADL provides an opportunity for establishing collaboration patterns tuned to team performance and quality metrics. Thus currently implicit best practises can be made explicit and subsequently shared. When customized to a particular platform such as MediaWiki, hADL provides a high-level view of the collaboration infrastructure. It thereby facilitates planning and documenting the platform's configuration and extensions.

[4] `http://www.eclipse.org/modeling/m2t/?project=jet#jet`

8 Conclusions

We made the case for a human Architecture Description Language for modeling adaptive collaboration structures. Taking inspiration from software architecture, we proposed hADL to specify collaboration patterns in terms of human components, collaboration connectors, and collaboration objects. A set of principles guides the design process to achieve collaboration patterns that facilitate runtime adaptation. Our evaluation successfully demonstrated that hADL supports the dynamic reconfiguration of human components and collaboration connectors at runtime. Nevertheless, even MediaWiki's adaptations capabilities are currently limited to the configuration of group permissions.

Our future work, therefore, will focus on the mapping between the underlying IT infrastructure and collaboration patterns. Ultimately, we aim for techniques that exploit the interdependencies between software elements and collaboration elements for achieving holistic co-adaptation of socio-technical systems. Such work will then also model and exploit diverse relationships between humans such as friendship, rivalry, dis/trust, and organizational hierarchy.

Acknowledgment. This work is supported in part by the National Science Foundation (CCF-0917129, CCF-0820222, and CCF-0808783) and the Austrian Science Fund (FWF) J3068-N23.

References

1. Ceri, S., Daniel, F., Matera, M., Facca, F.M.: Model-driven development of context-aware web applications. ACM Trans. Internet Technol. 7 (February 2007)
2. Chopra, A.K., Paja, E., Giorgini, P.: Sociotechnical Trust: An Architectural Approach. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 104–117. Springer, Heidelberg (2011)
3. Dashofy, E.M., van der Hoek, A., Taylor, R.N.: A comprehensive approach for the development of modular software architecture description languages. ACM Trans. Softw. Eng. Methodol. 14, 199–245 (2005)
4. Dorn, C., Taylor, R.N.: Analyzing runtime adaptability of collaboration patterns. In: International Conference on Collaboration Technologies and Systems (CTS). IEEE Computer Society, Los Alamitos (2012)
5. Dorn, C., Taylor, R.N., Dustdar, S.: Flexible social workflows: Collaborations as human architecture. IEEE Internet Computing 16, 72–77 (2012)
6. Dustdar, S.: Caramba- Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams. Distributed Parallel Databases 15(1), 45–66 (2004)
7. Garlan, D., Monroe, R., Wile, D.: Acme: an architecture description interchange language. In: Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON 1997, pp. 169–183. IBM Press (1997)
8. 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)
9. Little, G., Chilton, L.B., Miller, R., Goldman, M.: Turkit: Tools for iterative tasks on mechanical turk. In: Human Computation Workshop, HComp 2009 (2009)

10. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. ACM Comput. Surv. 26, 87–119 (1994)
11. Minder, P., Bernstein, A.: Crowdlang - first steps towards programmable human computers for general computation. In: Proceedings of the 3rd Human Computation Workshop (HCOMP 2011). AAAI Press (January 2011)
12. Moody, P., Gruen, D., Muller, M.J., Tang, J., Moran, T.P.: Business Activity Patterns: A New Model for Collaborative Business Applications (2006)
13. 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://public.dhe.ibm.com/software/dw/wes/1004_nandi/1004_nandi.pdf`
14. Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime software adaptation: framework, approaches, and styles. In: Companion of the 30th Intl. Conf. on Software Engineering, ICSE Companion 2008, pp. 899–910. ACM, New York (2008)
15. Schall, D.: A human-centric runtime framework for mixed service-oriented systems. Distributed and Parallel Databases 29, 333–360 (2011)
16. Schwinger, W., Retschitzegger, W., Schauerhuber, A., Kappel, G., Wimmer, M., Pröll, B., Castro, C.C., Casteleyn, S., Troyer, O.D., Fraternali, P., et al.: A survey on web modeling approaches for ubiquitous web applications. International Journal of Web Information Systems 4(3), 234–305 (2008)
17. Silva Souza, V.E., Lapouchnian, A., Mylopoulos, J.: System Identification for Adaptive Software Systems: A Requirements Engineering Perspective. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 346–361. Springer, Heidelberg (2011)
18. Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture: Foundations, Theory, and Practice. Wiley (2009)
19. Taylor, R.N., Medvidovic, N., Oreizy, P.: Architectural styles for runtime software adaptation. In: WICSA/ECSA, pp. 171–180 (2009)
20. Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., González, P.: CSRML: A Goal-Oriented Approach to Model Requirements for Collaborative Systems. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 33–46. Springer, Heidelberg (2011)
21. Wilde, E., Gaedke, M.: Web engineering revisited. In: BCS Int. Acad. Conf. pp. 41–50 (2008)
22. Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 161–176. Springer, Heidelberg (2006)