Analyzing Runtime Adaptability of Collaboration Patterns

Christoph Dorn  
Institute for Software Research  
University of California, Irvine  
cdorn@uci.edu

Richard N. Taylor  
Institute for Software Research  
University of California, Irvine  
taylor@uci.edu

Abstract—With the emergence of large-scale complex systems, the boundary between humans and software becomes increasingly blurry. Adaptation mechanisms thus need to consider the interaction structure amongst users to derive appropriate adaptation actions. Most existing systems, however, lack such collaboration awareness and thus cannot deliver the required flexibility that large-scale collaboration need to remain operational. In this paper, we analyze the flexibility of the underlying collaboration structures based on behavior, asynchrony, state, and execution context. The evaluation of three collaboration patterns and exemplary real world systems demonstrates that collaboration pattern awareness is of utmost importance for apprehending a system’s adaptation constraints and subsequently enacting appropriate adaptation actions.

Keywords—collaboration patterns, software architecture, evaluation framework, adaptation flexibility

I. INTRODUCTION

Over the past twenty years we have observed a trend towards "social software" leaving the realm of group support systems for small or medium sized teams and entering the dimension of internet-scale collaborations. Especially noteworthy is the increasingly blurry boundary between humans and software. Humans have become both provider and consumer of content and computation. Humans are no longer just the “users” of a system but an integral part [1]. Their interactions with other humans and software elements have a significant impact on the runtime management and adaptation of software components.

Research in adaptation mechanisms, however, has so far focused only on the software system and considered the implications arising from collaboration interdependencies as static requirements. A system neglecting the dynamic collective behavior, however, is unable to support efficient operation and evolution of collaborative efforts. Contemporary Web-scale systems for collaborative knowledge creation (e.g., Wikipedia), task crowdsourcing (e.g., Amazon Mechanical Turk), or information sharing (e.g., Twitter) lack interaction-aware adaptation capabilities and thus cannot provide but the simplest coordination forms. Harnessing the power of massive user participation for problems that require complex coordination thus remains out of reach.

As repeatedly pointed out [2], [3], software architecture-based approaches to self-management address adaptation on the right level of abstraction and generality, rather than focusing on language-level or network-level adaptation. On the architectural level, adaptation actions typically replace components and reconfigure connectors. To this end, the underlying architectural style determines to a large extent the effort required to implement runtime changes [4].

We argue that the same holds true for the human collaboration structure. Here, architecture-inspired adaptation describes changes in terms of who is executing work, who is coordinating the work, and how to (re)wire these collaborators. Along these lines, collaboration patterns at design-time define what aspects change, how long adaptation takes, what the side effects on the collaboration state are, and which adaptation restrictions exist. Patterns at runtime enable reasoning about the whole structure of the collaboration rather than focusing on individual interactions, collaborators, or shared artifacts. This motivates the main contribution of our paper: we revise the BASE framework [4] (initially for evaluating adaptation flexibility of software architectures) to discuss the adaptability aspects of collaboration patterns. Subsequently, we apply the framework to a set of collaboration patterns and exemplary existing systems. Our analysis highlights that selecting the appropriate collaboration patterns at design-time has a fundamental impact on system adaptability. Furthermore, adaptation mechanisms require explicit information about the underlying collaboration structures (in terms of collaboration connectors and human components) to arrive at the right, interaction-aware adaptation decisions at runtime.

The remainder of this paper is structured as follows: Section II provides a motivating scenario and outlines how multiple collaboration patterns interlock. Section III describes the BASE framework and its applicability to collaboration patterns. We subsequently apply the BASE aspects in Section IV for discussing the Shared Artifact, Master/Worker, and Publish/Subscribe pattern. For each of these patterns, we examine the flexibility of a corresponding real world, Internet-scale system in Section V. We summarize our findings in Section VI, provide related work in Section VII, and conclude with an outlook on future work in Section VIII.
II. Motivating Example

Monitoring and safety systems range in scope from a small security team handling an office building to thousands of personnel in back offices and on site at geographically distributed locations to secure critical infrastructure. These systems tightly interweave people and software components and hence need to consider the adaptation of collaboration structures to maintain overall system effectiveness and efficiency.

In the building monitoring case, back office operators utilize high definition video streams, floor plans, building sensor feeds, occupancy logs, and communication channels with on-site security staff. Staff of an earlier shift needs to provide observations, events and other data to team members of the next shift without necessarily knowing their exact identity. Collaboration flexibility is also required to seamlessly replace staff members, to temporarily bring in experts, or to reassign members to another subteam on the fly. Staff in sequential work shifts (and also within shifts) thus use a joint log book to avoid direct handover of information.

When back office personnel investigates suspicious behavior it has to gather a plethora of information to assure a certain situation is non-threatening. Analysis of raw data from multiple video feeds, still images, and voice recordings easily overwhelms a small team and thus requires additional staff members on demand. Teams need to coordinate analyst availability, task assignments, and task progress tracking. Adaptation actions such as replication and reassignment of users and analysis tasks aim to provide timely and high quality results.

In the case of a safety critical situation, adequate reactions require situation awareness. Staff members thus need continuous data about the location, equipment, and activities of their fellow staff members as well as system elements. Simultaneously, the need to provide such information about themselves to others. Individual staff members should not be occupied by trying to keep track of continuously fluctuating relevant users and system elements that represent information sources and sinks.

Coordination amongst the various users (and user groups) takes on multiple collaboration forms. We identify three potential collaboration patterns that satisfy the underlying coordination requirements: (1) the Log Book as a Shared Artifact, (2) Master/Worker for managing tasks, and (3) Event distribution through Publish/Subscribe (see Figure 1).

Each of these patterns has different idiosyncrasies when it comes to collaboration-level adaptations. Connecting two staff members, for example, corresponds to adding a link between two human components. The underlying pattern, however, has profound implications on whether or how this is possible. The Shared Artifact foresees only indirect communication by linking two users to the same artifact. The Master/Worker pattern establishes a link only when the Master assigns a task to a worker. Publish/subscribe supports unidirectional and bidirectional links as only one or both users subscribe to event updates. The next section introduces the BASE framework for analyzing the pattern-specific adaptability.

III. Approach

In their seminal paper, Malone and Crowston highlight the existence of similar coordination aspects in Economics, Organizational Theory, and Computer Science [5]. Just as a software architecture represents an abstraction of the technical system, so does a collaboration pattern describe the high-level structure of a collaborative effort. The basic elements in a software architecture are components and connectors. At any given level of abstraction, components are the loci of computation and data management whereas connectors coordinate the interactions between components. The distinction of connectors and components is mirrored in collaborations. We can distinguish between humans according to work-focused and coordination-focused roles. Roles such as managers, team leaders, secretaries are rarely described as connectors but they perform a similar task: the coordination of other humans (i.e., components).

As emphasized in [4], a set of common principles for adaptability can be extracted from the different architecture styles. The principles highlight enabling replaceability of elements that are intended to change and controlling interactions with the dynamic elements. Similarly, we would like to achieve adaptability in large-scale collaborations. One of the main purposes of a Collaboration Connector is decoupling of Human Components. In the Master/Worker pattern, the task creator (Master) and task executer (Worker) comprise the human components. Without a collaboration connector, Master and Worker need to be aware of each others identity to get in direct contact for the various interaction stages such as tasks matching, task allocation, progress monitoring, worker replacement, etc. Here, a collaboration connector provides vital management capabilities to simplify those interactions. Note that a collaboration connector is often a piece of software and thus not actually implemented by a user (or a heavily software supported user). Note that we, therefore, make no distinction between human initiated actions and software initiated actions when discussing a collaboration pattern.

As the scenario demonstrates, complex mixed systems combine multiple collaboration patterns. Human targeted adaptation mechanisms (e.g., recommendations, filtering, (semi)-
automatic reconfigurations) thus need to become aware what scope of reconfigurations these patterns support. We thus propose to treat collaborations in terms of human components and collaboration connectors and subsequently refashion tools from the software architecture domain for analyzing a pattern’s adaptability.

A. Software Runtime Adaptation Aspects

The BASE framework [4] defines four aspects relevant for software runtime adaptation: behavior, asynchrony, state, and execution context. We revisit the initial BASE aspects and their significance for runtime software adaptation and then outline how these properties are equally applicable for analyzing collaboration patterns.

Behavior highlights the scope of supported change. This aspect concerns the means for changing a system’s behavior and the respective level of abstraction (e.g., reconfiguration at code level or at component level). Some styles limit adaptations to compositions of existing behaviors while others allow the introduction of new behavior. Limitations also include specification of behavior that must remain unmodified.

Asynchrony addresses the implications that come with the lag between initiating a system’s adaptation and its completion. Large-scale distributed systems take potentially longer to update than compact, central systems and might never reach a completely updated status. This aspect also includes maintenance of system constraints during adaptation and continuous system availability.

State refers to potential adaptation “side effects” that have an impact on the system’s state. Changing a data type definition might require updating all current instances to the new definition. Replacement of a component potentially involves extracting that component’s state for initializing the new component before the system can resume.

Execution Context raises awareness of constraints that determine whether or not adaptation can commence. For example, a component currently having control cannot be modified directly.

These aspects represent equally important concerns in the context of collaboration patterns.

B. Collaboration Runtime Adaptation Aspects

Compositions from human components and collaboration connectors produce patterns similar to architecture styles. Given the similarities highlighted above, the BASE framework applied to collaborations consists of the same runtime adaptation aspects. Note that part of this paper derives from a technical report [6] and we also demonstrate the applicability of BASE in the scope of web-scale workflows in [7].

Behavior similarly addresses the means for adaptation. Reconfiguration of collaboration patterns does not stop at changing a single user’s involvement. Again, there are multiple levels of granularity that allow behavior adaptation. Replacement of a complete company, a team, adding another worker, or acquiring a required skill all correspond to component-level adaptation in software systems. Overall, adaptation goes beyond human components and connectors. Also messages between members and shared artifacts represent significant loci of adaptation.

Asynchrony in collaborations refers to the time it requires to establish a (new) team, replace a worker, become acquainted to another worker, or learn a new skill and how the joint work is affected during that phase. This aspect likewise raises awareness on constraints that need to be enforced during the reconfiguration. For example, when replacing a complete team, at least one former member must remain available during the transition phase (rather than exchanging all members at once).

State aspects draw attention to direct and indirect adaptation side effects when altering the means of communication, the manipulation of shared artifacts, or replacement of workers. The most knowledgable form of direct state change is loss of implicit collaboration know-how upon removing a worker. Handover of such implicit collaboration information between outgoing and incoming workers needs explicit consideration when adapting the human interaction structure.

Execution Context refers to the possibility to adapt during an active collaboration session. Whether a human may cease work on a particular task, or whether it is necessary to wait until task completion depends on multiple factors such as explicit contracts, cost and time for repeating the task, or executing compensation actions. The same holds true for the degree of coupling between two or more workers during an ongoing interaction.

IV. APPLYING BASE FOR COLLABORATION PATTERNS

Numerous collaboration patterns exist to address various management concerns such as coordination of (i) shared resources, (ii) producer/consumer relationships, (iii) simultaneity constraints, and (iv) task/subtask relations [5]. In this section, we will focus on three popular patterns from our scenario: (1) Shared Artifact, (2) Master/Worker, and (3) Publish/Subscribe.

For each pattern we propose a matching software architecture style followed by a longer discussion with BASE. The juxtaposition of the software style and collaboration pattern enables us to better perceive the similarities and thus to gain an improved understanding of the potential directions for human collaboration adaptation.

A. Shared Artifact

Tuple spaces are a close match for Shared Artifacts in the software engineering domain. They provide strong decoupling of components as interactions are limited to publicly visible manipulations of data items. A tuple-space manages storage and retrieval of data items (i.e., artifacts) thereby imitating a (distributed) shared memory [8]. A component obtains a lock, manipulates the shared artifact locally, pushes the update to the space, and releases the lock again. In the mean time, others maintain read access to the previous version.

A Shared Artifact describes any type of activity where participants communicate predominantly through the manipulation of a common object such as a document, source code, or a discussion board (Fig. 2). Research in the domain of
CSCW focused early on such capabilities in the context of shared workspace systems and groupware systems [9]. Several approaches were proposed for large-scale environments until Internet-scale collaborative editing became a universal success with the emergence of Wikis [10]; the most prominent example being Wikipedia.

Behavior: Adaptation actions consist of adding/removing collaborators to/from artifacts, moving collaborators between artifacts, improving on the artifact type to convey more coordination-enabling information, and artifact structuring (splitting/merging/replication). In self-organizing environments, collaborators may join or leave the workspace at any time. They are free to create new shared artifacts or manipulate existing ones without requiring a dedicated person collecting, merging, and distributing contributions. Collaboration connectors govern artifact access privileges, and thus determine the need to detect, prevent, resolve, and decide upon change conflicts.

Asynchrony: Adaptation actions that occur on a per artifact basis affect only a subset of all collaborators. They need instructions on how to deal with the simultaneous existence of old and new artifact types as shared artifacts are incrementally updated. In contrast, demanding a synchronous type update of all artifacts at once prevents every collaborator from performing artifact manipulations for the duration of the complete system adaptation.

State: The shared artifact maintains the collaboration’s state. Collaborators construct their internal state from the artifact’s history without having to contact all involved participants individually. Currently active collaborators thus need not transfer state to future collaborators which may not be known in advance.

Execution Context: When upgrading an existing artifact (e.g., splitting a large document into individual chapters), all write access requests need to wait while the upgrade takes place. In return, adaptation actions need to wait for collaborators to complete their artifact manipulation activities. Small, self-contained changes allow for timely artifact adaptations. Locking mechanisms, however, need to be in place when collaborators tend to update large parts that likely lead to conflicts.

B. Master/Worker

Master/Worker and its software counterpart Map-Reduce leverage parallel computing. Map-Reduce divides a task into multiple independent subtasks [11]. A central coordinator node subsequently assigns each subtask to a different processing machine (map phase). In the reduce phase, the coordinator collects the individual task outcomes and distributes them again for aggregation to generate the final result.

Partitioning tasks also works well in human collaboration when the resulting work items remain independent (Fig. 3). The master defines the individual work packages. In contrast to Map-Reduce, task assignment occurs in push and pull style. The former procedure has an Assignment Connector allocate tasks directly to workers, whereas the latter procedure enables workers to choose which task they prefer to work on. Another difference lies in the reduce phase. The assignment connector merely collects the results and relays them back to the master who then has to aggregate the individual results.

The large-scale deployment of the Master/Worker pattern is often referred to as crowdsourcing [12], the most prominent example being Amazon Mechanical Turk.

Behavior: The master decides upon the number of workers that can work in parallel on copies of the same task artifact. In the pull-style assignment, workers choose which tasks to perform, and whether to return a task unfinished. In push-style assignment, workers receive new tasks in their job queue but may still have the option to reject or delegate a task.

Asynchrony: A task artifact completely decouples master and workers. The master, respectively the assignment connector, has the option to reassign the task to another worker (or make it available again) when the worker fails to complete the task in a predefined time-frame. The Master informs the Assignment Connector on the number of multiple identical tasks for sake of reliability, or the issuing of multiple sequential tasks until the desired result quality has been achieved.

State: The task artifact contains the complete collaboration state. Replacement of workers has no side-effect on the state.

Execution Context: All workers execute their task independently, hence, no synchronization of results is required. Multiple workers assigned to the same task artifact work on distinct copies and have no knowledge about each other. They remain similarly unaware of any replacement of the master. A new worker simply obtains the task description and commences task execution independently of any previous work done.
C. Publish/Subscribe

In the Publish/Subscribe style, publishers and subscribers communicate indirectly by means of events and are typically unaware of each other’s identity [13]. A message oriented middleware (e.g., a message bus) manages event collection and distribution. Subscribers process the received events and potentially produce new events of their own. Such strong component decoupling enables simple adaptation through runtime adding and removing of event producers and consumers [14].

In collaborative environments, mailing lists (e.g., List-serv [15]) emerged among the earliest instances of pub/sub to push information of general interest to a larger audience. This collaboration pattern comes in different flavors characterized by the anonymity of sender and receivers, the ability of receivers to reply or post their own message, and whether the list is topic or person-centric (Fig. 4). In recent years, microblogging platforms such as Twitter have become a popular tool for rapidly disseminating information in large-scale environments [16]. In contrast to purely topic-centric subscriptions in pub/sub, subscriptions in micro-blogging environments are mostly person-centric.

![Publish/Subscribe pattern example](image)

**Behavior:** With topic-centric lists, publishers and subscribers may dynamically join and leave while the respective list remains unaffected. The lifetime of a person-centric list usually remains coupled to the publishing activity of its respective author. Simultaneously, new topics emerge and existing topics lose their relevance.

**Asynchrony:** Changes to a list’s subscriber base have no side-effects. Single changes are instantaneous and require no synchronization with other users. Author removal from person-centric lists and single-publisher lists may cause receivers to resort to alternative event sources. However, support mechanisms for subscribing to relevant lists is outside the pattern’s scope.

**State:** The collaboration domain defines the requirements for state transfer upon swapping publishers. Consumers build their internal state as new messages arrive. Most mailing list and micro-blogging implementations maintain a history of past messages that enable immediate state reconstruction.

**Execution Context:** Message dispatching and receiving is considered atomic. Hence, independent unidirectional messages pose no restrictions on the replacement of publishers or readers. When multiple individual messages from multiple authors create a discussion thread the removal of an involved publisher requires other authors to compensate.

V. Example Systems

In the following subsections, we present three real world examples of the introduced collaboration patterns and analyze how the underlying interaction structure influences the support for adaptation.

A. Wikipedia

Wikipedia \(^1\) is one of the largest cooperative knowledge creation effort on the Internet [10]. Anybody may create a new article or improve on an existing one. The underlying Wiki technology keeps track of changes and thus allows to revert to any previous article version. An article typically links to other articles and external sources to validate its content. Wikipedia’s coordination capabilities go beyond the pure shared artifact pattern and come with additional mechanisms to handle vandalism, editing wars, article relocation, and voting on member privileges.

**Behavior:** The majority of articles are available for modification by regular users. Articles subject to vandalism or editing wars may become semi-restricted to confirmed editors. Regular users then have to explicitly request changes. Elected by their peers, some users become administrators (Collaboration Connectors) and obtain additional capabilities to remove articles, block users, and modify fully protected articles. While content updates have only local effect (those that read and update the article), updates to the Wiki metadata schema requires locking down the whole system for the duration of the database update. Consequently, such adaptation is very rare.

**Asynchrony:** Wikipedia allows simultaneous article changes in the absence of a locking mechanism. Editors are advised to submit small and frequent updates, and if possible, limited changes to a subsection but not the complete article. An automatic diff tool (Collaboration Connector) merges non-overlapping changes. Conflicting changes require manual merging by the last submitting editor. Alternatively, the `inuse` tag signals long-running article changes.

**State:** Each Wikipedia article lists its edit history on a separate web page. Each change entry details the responsible editor, the amount of change, a comment, and provides access to the historical article including differences to previous and current version.

**Execution Context:** Wikipedia provides a discussion page associated with each article that allows the negotiation of larger changes and/or solving of disputes. The main article remains untouched until the involved editors have reached an agreement about the content changes (Collaboration Connector).

Quality control is an important issue for Wikipedia. Each Wiki page thus allows editors to receive notifications (Collaboration Connector) about article changes, thereby simplifying the detection of vandalism and edit wars.

\(^1\)http://en.wikipedia.org/wiki/Wikipedia:About
B. Amazon Mechanical Turk

Amazon Mechanical Turk (MTurk) is one of the most prominent examples for crowdsourcing. The MTurk platform implements the Master-Worker collaboration pattern, enabling Requestors (i.e., Masters) to register a Human Intelligence Task (HIT). Workers are subsequently able to search through the set of registered HITs and accept HITs they are interested in. Once the worker has accepted a HIT, he has the allotted time (duration) to complete his copy of the task (i.e., his assignment). The requester retrieves the submitted assignment and approves it when the result quality is satisfactory, or otherwise rejects it without paying the worker.

**Behavior:** MTurk supports only pull-based task assignment and features a dynamic workforce that cannot be directly controlled. All adaptation actions unfold their effects through HIT manipulation.

Requestors have full control over the creation, reconfiguration, and removal of HITs. They regulate the wiring of workers to HITs through (i) direct blocking of workers and (ii) specification of required skills. The requestor defines HIT relevant skills and can assign/revoke the corresponding worker qualification anytime.

Workers have less control over HITs per se, but typically filter HITs for matching qualifications, rewards, and duration. In MTurk, only workers can establish a collaboration by accepting an assignment. Although they are able to return or abandon a HIT, this reflects negatively on their reputation.

**Asynchrony:** The crowdsourcing environment of MTurk does not foresee explicit replacement of a unresponsive or low quality worker. In the former case, MTurk automatically makes the HIT available again once the work duration has expired (Collaboration Connector). In the second case, the requestor needs to register the HIT again. The requester decides on appropriate HIT values for the number of simultaneous workers, reward, and duration to promote timely and high-quality results. Unattractive rewards or unrealistic timelines coupled with a lack of backup workers leads to delays in HIT execution.

Similarly, MTurk addresses unresponsive requestors. Each HIT specifies a deadline after which a submitted assignment is automatically approved.

**State:** Workers have read-only access to the HIT description. They also remain unaware of any other workers having (currently or previously) accepted the respective HIT. Returning or abandoning a HIT has thus no side-effect on any other worker assigned in parallel. The assignment result remains with the worker until he explicitly submits his work. No modifications are possible thereafter. The requester obtains read access to an assignment’s result as soon as it is submitted. Approval and rejection are independent of other assignments of the same HIT but multiple results are usually compared for quality assurance purposes.

**Execution Context:** The collaboration between requestor and worker ends upon assignment approval or rejection. Both parties can prematurely terminate a HIT anytime. A worker can return or abandon the HIT with reputation side-effects. The requestor has the options to force expire the HIT such that it becomes invisible to new workers but allows existing assignments to be completed. The result is then open for approval or rejection. Alternatively the requestor can disable a HIT which removes the HIT and automatically approves all submitted and pending assignments.

C. Twitter

Twitter is a micro-blogging platform used to create and maintain a social network as well as distributing news. Twitter supports collaboration according to the Publish/Subscribe style: every user is simultaneously a publisher and subscriber. A user subscribes to another user’s twitter feed, thereby becoming a follower of the other user.

When a user posts a message — denoted a tweet which may be up to 140 characters — it becomes visible on her profile page (if the account is public). At the same time, the tweet is pushed to the home timeline of all her followers. Twitter supports message meta information in the form of hash tags as topic descriptors, references to other users (so-called mentions, e.g., @BarackObama), and URL shorteners to link to external resources. Users apply retweeting to refer to a previous messages, or replies to respond to another user.

**Behavior:** Run-time behavior adaptation focuses on managing subscriptions and on message content. Subscribers are free to follow any publisher with a public profile. Subscription requests for private profiles need the publisher’s approval. Twitter provides a list of followers to each publisher. The publisher has the option to block a particular subscriber. This prevents messages from showing up on the subscriber’s home timeline, but they remain available to everyone on the publisher’s public profile.

A publisher typically distinguishes between different message types using hash tags. New hash tags may be introduced anytime. Users then search for a particular hashtag and subscribe to the respective tweeter feed to become a follower.

**Asynchrony:** An unavailable publisher has no direct effect on her subscribers. Followers merely cease to receive events from that publisher but continue to receive events from other publishers. Offline followers receive the latest events upon reconnecting to Twitter. They are not guaranteed to receive all messages after an extended duration of absence as Twitter does not provide the complete history but only the recent messages (Collaboration Connector). The subscription operation is typically instantaneous and new messages are accessible immediately. Messages are delayed until approval when requesting to follow a user with private profile.

**State:** Users gather information (i.e., state) from public profiles available before following the publisher. Hence, a user having just joined Twitter can easily perceive which existing users to follow. On the publisher’s side, messages remain available even when there are no followers.

\[^{2}\text{http://www.mturk.com}\]

\[^{3}\text{http://twitter.com}\]
Execution Context: There are no constraints for (un)following a publisher or blocking a subscriber. Merely private profiles require approval before a relation is established.

VI. Discussion

Choosing the right collaboration patterns has uttermost impact on the runtime adaptation flexibility. The log book as a shared artifact relieves staff of the earlier shift of directly knowing which users will be accessing the information next (Fig. 1 (1)). In return, staff in a later shift need not request such log information from their predecessors directly.

The task-like nature of information analysis highlights the Master/Worker pattern as a viable solution to manage the load on the core monitoring team and dynamically crowd source the content analysis (Fig. 1 (2)). Existing real world applications apply crowdsourcing for determining image content\(^4\) or transcribing recordings\(^5\). For monitoring purposes, we envision also language translation services and even on-site user observations.

Situation awareness through event distribution follows most often the Publish/Subscribe pattern. Individual staff members subscribe to relevant topic-based and person-based events (Fig. 1 (3)). A user then simply publishes an event without having to manage interested receivers. An additional benefit over typical software based Publish/Subscribe systems is awareness of interested subscribers. This enables users to establish reciprocal subscriptions.

The right underlying collaboration patterns in place is a prerequisite for realizing a runtime collaboration-level adaptation mechanism. The scenario’s monitoring system must handle internal failures (e.g., mitigating a non-responding team member) as well as external forces (e.g., reacting to unforeseen situations). To this end, awareness of the underlying collaboration patterns reveals the appropriate adaptation actions and corresponding implications.

All three patterns exhibit strong decoupling of collaborators. An adaptation mechanism is thus able to provide reconfiguration actions (e.g., in the form of recommendations) that affect only a subset of the overall team. A staff member receives a link to another subteam’s log book. In the crowdsourcing case, the task executing worker won’t notice the replacement of the task initiator. Subscription recommendations highlight relevant new staff members to add, and vice versa. In all these examples, the adaptation mechanism would not be able to perceive the right way to execute an adaptation action without knowing the pattern specific semantics. Actions to remove a member differ in each pattern: (1) revoking access to the shared artifact, (2) removing and reassigning tasks, and (3) dropping subscriptions, respectively. The analysis with BASE also highlights the differences in who to address for adaptation, when to adapt, and how long that adaptation will take.

The presented real world systems demonstrate pattern scalability and flexibility but they lack specific capabilities required for the scenario application. Upgrading the patterns with additional functionality without breaking the underlying interaction structure calls for the replacement and/or deployment of collaboration connectors:

A log book needs a connector that manages access rights based on various user properties. As software elements provide particular log entries, we cannot expect them to detect and resolve write conflicts as done in Wikipedia. A possible solution might foresee a dedicated locking mechanism just for these software elements.

Crowdsourcing of information analysis requires additional steps beyond what MTurk offers to break down tasks into subtasks, to ensure result quality, and aggregation of results [18]. Again, this is the purpose of connectors who collect results and submit new tasks without having the initial Master involved.

Twitter focuses on people-centric subscriptions and thus lacks capabilities to subscribe to topics or contextual conditions. We suggest a collaboration connector to dynamically recommend relevant team members (or even automatically create and remove subscriptions) based on task, location, or organizational proximity. Alternatively, a connector might apply those criteria to filter or rank events.

Limitations

Patterns per se cannot guarantee successful collaboration structure adaptation. Human factors such as social ties, previous interactions, and trust play a major role when deciding upon specific adaptation details. Independent of these properties, however, the underlying pattern — through its implementing IT infrastructure — provides a set of constraints that determine the scope of adaptation actions. The pattern determines the relevance of the human factors. The Master/Worker pattern, for example, emphasizes worker skills and reliability rather than trust amongst workers. In contrast, subscribers will trust information from known publishers more than information from newcomers. Summarizing, we are well aware of the importance of human factors and we strongly suggest to consider them for planning specific adaptation actions (e.g., which particular communication ties to establish, which human to add, what artifacts to recommend). We, however, must stress that such factors cannot properly be applied without understanding the pattern’s implications.

VII. Related Work

Related research efforts are scattered across multiple scientific fields. Most research focuses on individual collaboration, respectively coordination, patterns. Crowdsourcing of human tasks has received significant attention in the last years. Skopik et al. investigate interaction patterns for trust-based routing of tasks amongst interconnected experts [19]. Kittur et al. outline how the Map-Reduce architectural style enables crowdsourcing of complex workflows [18]. Dusdstar and Gaedke propose context-dependent selection of collaboration structures for task delegation [20]. Schall et al. propose the Broker Query Description Language to detect experts that bridge
separate communities [21]. Analysis of message flows in twitter exposes properties of large-scale social Publish/Subscribe systems [16]. Observations of twitter messages also show how this pattern is applicable to social networking purposes as well as information dissemination [17]. Team automata aim to formalize the interactions amongst multiple participants in groupware systems [22]. Their nature, however, lend them more to the analysis and design of access security mechanisms rather than reasoning about collaboration adaptivity.

Amongst the few interdisciplinary works, Malone and Crowston highlight the existence of similar coordination aspects in Economics, Organizational Theory, and Computer Science [5]. Similarly, [23] and [24] discuss general aspects of large-scale collaboration systems but don’t address the corresponding adaptation flexibility. Malone, Laubacher, and Dellarocas [24] characterize crowdsourcing systems by distinguishing between crowd-based and hierarchy-based actions, the actions themselves (create or decide), motivation (money, love, or glory), and propose generic collaboration forms. In a similar attempt, Doan, Ramakrishnan, and Halevy [23] analyze whether collaboration is explicit or implicit, relies on existing data, what users do, and give example real world systems. Yet to the best of our knowledge, this is the first attempt to analyze adaptation flexibility across collaboration patterns.

VIII. CONCLUSIONS

Adaptation of large-scale collaboration systems relies on the appropriate underlying patterns and awareness how these influence the system’s flexibility during runtime. We propose to apply the BASE framework to evaluate behavior, asynchrony, state, and execution context of collaboration patterns in order to reason about the range of suitable adaptation actions. The study of three collaboration patterns and exemplary real world systems demonstrates the benefits of BASE to assess the compatibility of collaboration patterns and intended use.

Our future work involves the study of additional collaboration structures such as secretary/principal, organizational control, workﬂows, and social networks. We will further analyze the horizontal integration of various patterns as suggested in [20]. To this end, we have started work on a human architecture modeling language that allows the speciﬁcation of human components, collaboration connectors, and their conﬁguration for specifying various collaboration patterns.

REFERENCES

[1] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, “Ultra-Large-Scale Systems - The Software Challenge of the Future,” Software Engineering Institute, Carnegie Mellon, Tech. Rep., June 2006.
[2] P. Oreizi, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, “An architecture-based approach to self-adaptive software,” IEEE Intelligent Systems, vol. 14, pp. 54–62, May 1999.
[3] J. Kramer and J. Magee, “Self-managed systems: an architectural challenge,” in International Conference on Software Engineering, 2007, pp. 259–268.
[4] R. N. Taylor, N. Medvidovic, and P. Oreizy, “Architectural styles for runtime software adaptation,” in WICSAECSA, 2009, pp. 171–180.
[5] T. W. Malone and K. Crowston, “The interdisciplinary study of coordination,” ACM Comput. Surv., vol. 26, pp. 87–119, March 1994.
[6] C. Dorn and R. N. Taylor, “Mapping software architecture styles and collaboration patterns for engineering adaptive mixed systems,” Institute of Software Research, University of California, Irvine, http://www.isr.uci.edu/tech_reports/UCI-ISR-11-4.pdf, Tech. Rep., June 2011.
[7] C. Dorn, R. N. Taylor, and S. Dustdar, “Flexible social workﬂows: Collaborations as human architecture,” IEEE Internet Computing, vol. 16, pp. 72–77, 2012.
[8] D. Gelernter, “Generative communication in linda,” ACM Trans. Program. Lang. Syst., vol. 7, pp. 80–112, January 1985.
[9] C. A. Ellis and S. J. Gibbs, “Concurrency control in groupware systems,” in Proceedings of the 1989 ACM SIGMOD international conference on Management of data. ser. SIGMOD ’89. New York, NY, USA: ACM, 1989, pp. 399–407.
[10] A. Kittur and R. E. Kraut, “Harnessing the wisdom of crowds in wikipedias: quality through coordination,” in Proceedings of the 2008 ACM conference on Computer supported cooperative work. ser. CSCW’08. New York, NY, USA: ACM, 2008, pp. 37–46.
[11] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Commun. ACM, vol. 51, pp. 107–113, January 2008.
[12] D. C. Brahmab, “Crowdsourcing as a model for problem solving,” Convergence: The International Journal of Research into New Media Technologies, vol. 14, no. 1, pp. 75–90, 2008.
[13] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The many faces of publish/subscribe,” ACM Comput. Surv., vol. 35, pp. 114–131, June 2003.
[14] L. Fiege, F. C. Gätter, O. Kasten, and A. Zeidler, “Supporting mobility in content-based publish/subscribe middleware,” in Middleware, 2003, pp. 103–122.
[15] D. Grier and M. Campbell, “A social history of bitnet and listserv, 1985–1991,” Annals of the History of Computing, IEEE, vol. 22, no. 2, pp. 32–41, apr-jun 2000.
[16] D. Zhao and M. B. Rosson, “How and why people twitter: the role that micro-blogging plays in informal communication at work,” in Proceedings of the ACM 2009 international conference on Supporting group work, ser. GROUP ’09. New York, NY, USA: ACM, 2009, pp. 243–252.
[17] H. Kwak, C. Lee, H. Park, and S. Moon, “What is twitter, a social network or a news media?” in WWW ’10: Proceedings of the 19th international conference on World wide web. New York, NY, USA: ACM, 2010, pp. 591–600.
[18] A. Kittur, B. Smus, and R. Kraut, “CrowdForge: crowdsourcing complex work,” in Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems, ser. CHI EA ’11. New York, NY, USA: ACM, 2011, pp. 1801–1806.
[19] F. Skopik, D. Schall, and S. Dustdar, “Trusted interaction patterns in large-scale enterprise service networks,” in Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, ser. PDP ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 367–374.
[20] S. Dustdar and M. Gaedke, “The social routing principle,” Internet Computing, IEEE, vol. 15, no. 4, pp. 80–83, july-aug. 2011.
[21] D. Schall, F. Skopik, H. Psenner, and S. Dustdar, “Bringing socially-enhanced virtual communities,” in Proceedings of the 2011 ACM Symposium on Applied Computing, ser. SAC ’11. New York, NY, USA: ACM, 2011, pp. 792–799.
[22] C. Ellis, “Team automata for groupware systems,” in Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge. ser. GROUP ’97. New York, NY, USA: ACM, 1997, pp. 415–424.
[23] A. Doan, R. Ramakrishnan, and A. Y. Halevy, “Crowdsourcing systems on the world-wide web,” Commun. ACM, vol. 54, pp. 86–96, Apr. 2011.
[24] T. W. Malone, R. Laubacher, and C. Dellarocas, “Harnessing crowds.: Mapping the genome of collective intelligence,” Technology, vol. 1, no. 2, pp. 327–335, 2010.