CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2015; **27**:2725–2750
Published online 28 November 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3438

SPECIAL ISSUE PAPER

Analyzing runtime adaptability of collaboration patterns

Christoph Dorn[1,*,†] and Richard N. Taylor[2]

[1]*Distributed Systems Group, Vienna University of Technology, Vienna, Austria*
[2]*Institute for Software Research, University of California, Irvine, CA, USA*

SUMMARY

The recent two decades have witnessed the emergence of large-scale, interaction-intensive systems. A system's provided user-centric communication and coordination mechanisms have a significant impact on its runtime management. Beyond a certain size, manual monitoring and management are no longer feasible. Hence, it is highly important for a system designer to becoming aware of the most suitable interaction mechanisms and their implications on system adaptability. Specifically, a system designer requires knowledge on what adaptation primitives are available, whether these are system-driven or user-driven, how long they will take, what impact do they have on collaboration state, and under what conditions they can be enacted. These aspects vary considerably across collaboration patterns.

In this paper, we investigate a collaboration structure's adaptability based on behavior, asynchrony, state, and execution context. We subsequently discuss seven distinctively different collaboration patterns in terms of those aspects. Based on a motivating scenario, we ultimately demonstrate how these patterns and insights into their inherent adaptability may guide design decision impact and trade-off analysis. Copyright © 2014 John Wiley & Sons, Ltd.

Received 19 February 2014; Revised 17 September 2014; Accepted 5 November 2014

KEY WORDS: collaboration pattern; software architecture; evaluation framework; adaptation flexibility; interaction-intensive systems

1. INTRODUCTION

Over the past 20 years, we have observed a trend toward 'social software' leaving the realm of group support systems for small-sized or medium-sized teams and entering the dimension of Internet-scale collaborations. We find collective intelligence and user participation among the main characteristics of the Web 2.0 [1]. 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 [2]. Their interactions with other humans and software elements have a significant impact on the system's runtime management and thus require dedicated consideration during the system's design.

A software design team tasked with devising the key architectural decisions for an interaction-intensive system (e.g., monitoring critical infrastructure) needs to decide on—among other things—the potential ways the various end users will eventually communicate. Being aware of the applicable interaction mechanisms' idiosyncrasies is highly important. First, the appropriate interaction mechanisms determine the underlying system's ability to effectively and efficiently fulfill its purpose by establishing the desired balance between (1) system control over user actions

*Correspondence to: Christoph Dorn, Distributed Systems Group, Vienna University of Technology, Argentinierstrasse 8/184-1, 1040 Vienna, Austria.
†E-mail: dorn@dsg.tuwien.ac.at

and (2) system-facilitated user action flexibility. Second, they also determine a system's ability for adaptation at runtime. Regardless whether a system tends toward rigid control of user actions or allows a large degree of interaction freedom, once a system acquires a considerably large and complex user base, automatic management capabilities need to be in place. Beyond a certain size, manual monitoring and management are no longer feasible. Designing such automatic capabilities hence requires insights into an interaction structure's runtime adaptability. Crucial properties include what adaptation primitives are supported, whether these are system-driven or user-driven, how long they will take to complete, what impact they have on collaboration state, and under what conditions they can be enacted.

The effects of lacking insight into interaction flexibility, respectively adaptability, are twofold. First, designers having to choose between two alternative interaction structures remain unaware of trade-offs. The resulting system ends up brittle as, for example, a rigid interaction mechanism (e.g., a workflow system) is enhanced with complex features to provide additional flexibility rather than merely adding restrictions to an intrinsically flexible mechanism (e.g., an instant messaging system). Second, designers oblivious to a particular interaction structure's side effects might arrive at a system design that will not be as adaptable at runtime as expected. For example, having an adaptation mechanism requires simultaneous availability of all collaborators for upgrading a conference call software will remain potentially unavailable for a considerable time.

In our view, one aspect in addressing this shortcoming consists of explicitly modeling and analyzing user interaction structures. Typical software development activities focus on abstracting the technical aspects: component and connector architecture view, action sequences, data structures, data flow, subsystem distribution, and so on. Little effort is spent on abstracting interaction structures among users (besides the rather limited UML use case diagrams). Especially, design approaches for runtime adaptation mechanisms have primarily focused only on the software system [3] and have taken the implications arising from collaboration interdependencies as static requirements [4].

Software-centric engineering approaches, however, hold the key to the solution. As repeatedly pointed out [5, 6], software architecture-based self-management techniques 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 [7].

We argue that the same holds true for the users' collaboration structure. In the case of interactions, architecture-inspired adaptation describes changes in terms of who is executing work, who is coordinating the work, and how to (re)wire the collaborators. Furthermore, collaboration structures exhibit patterns similar to architectural styles. Taylor, Medvidovic, and Oreizy define architectural styles as follows [8] p.73:

> An *architectural style* is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.

Example software architectural styles include *Pipe-and-Filter*, *Blackboard*, *Representational State Transfer*, and *Peer-to-Peer*. Collaboration patterns provide the same benefit when designing user interaction structures. Specifically, collaboration patterns allow the design team to reason about what aspects change, how long adaptation takes, what the side effects on the collaboration state are, and which adaptation restrictions exist. The design team can then devise mechanisms that observe these patterns at runtime for better monitoring, analyzing, and adaptation planning of the overall collaboration structure rather than focusing on individual interactions, collaborators, or shared objects.

This motivates the main contribution of our paper. We focus on analyzing collaboration adaptability (including interaction flexibility and adaptation control) while leaving the issue of formally representing collaboration structures to another paper (see e.g., [9]). To this end, we revise the Behavior, Asynchrony, State, and Execution (BASE) framework [7] (initially contrived for evaluating runtime adaptability of software architectures) for discussing the adaptability aspects

DOI: 10.1002/cpe

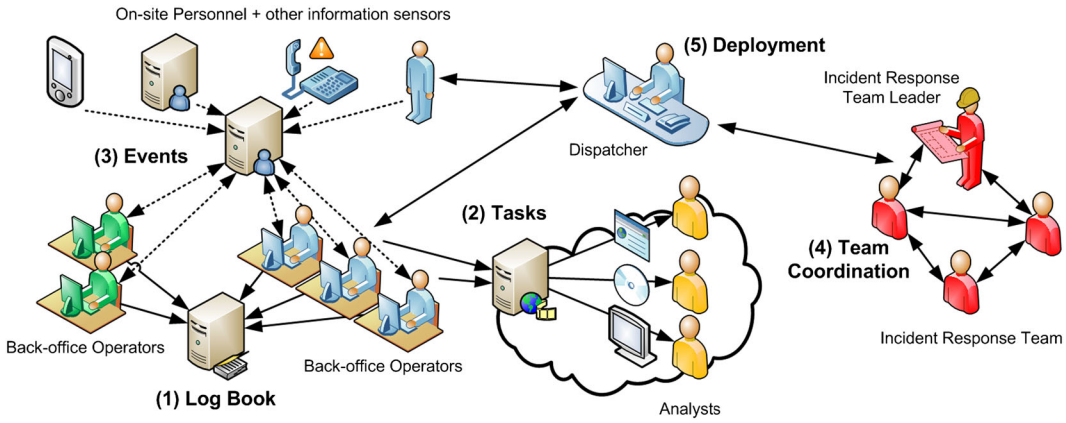

Figure 1. Collaboration aspects in an infrastructure-monitoring scenario.

of collaboration patterns.[‡] We subsequently apply the revised framework to the following seven collaboration patterns: *Shared Artifact* (4.1), *Publish/Subscribe* (4.2), *Master/Worker* (4.3), *Social Network* (4.4), *Workflow* (4.5), *Secretary/Principal* (4.6), and *Organizational Control* (4.7). Our analysis highlights the fundamental differences in adaptability. Based on a motivating scenario, we demonstrate how our contribution assists a design team in evaluating trade-offs and arriving at informed, optimal design decisions.

The remainder of this paper is structured as follows: Section 2 provides a motivating scenario and outlines a variety of crucial collaboration pattern-centric design decisions. Section 3 describes the BASE framework and its applicability to collaboration patterns. We subsequently apply the BASE aspects in Section 4 for discussing the previously listed seven patterns. We demonstrate in Section 5 how these patterns and insights into their inherent adaptability may guide design decision impact and trade-off analysis. We discuss our findings in Section 6, provide related work in Section 7, and conclude with an outlook in Section 8.

2. MOTIVATING EXAMPLE

Monitoring and safety systems range in scope from a small security team handling an office building to thousands of operators 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 user flexibility within various collaboration structures to maintain overall system effectiveness and efficiency.

In the building monitoring case (Figure1), back-office operators utilize high-definition video streams, floor plans, building sensor feeds, occupancy logs, and communication channels with on-site security personnel. 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 operators, to temporarily bring in experts, or to reassign members to another subteam on the fly.

When back-office operators investigate suspicious behavior, they have 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 back-office team and thus requires additional analysts on demand. Teams need to coordinate analyst availability, task assignments, and task progress tracking.

[‡]Note that this paper is a considerably extended version of [10] and derives from a technical report [11]. We also briefly demonstrated the applicability of BASE in the scope of web-scale workflows in an invited IEEE Internet Computing article [12].

DOI: 10.1002/cpe

In the case of a safety critical situation, adequate reaction requires situation awareness. Operators thus need continuous data about the location, equipment, and activities of their fellow workers as well as system elements. Individual operators should not be occupied by trying to keeping track of continuously fluctuating relevant users and system elements that represent information sources and sinks.

On site, incident response teams need communication mechanisms, which reflect the team's organizational structure as well as allow for flexible reaction to unforeseen situation development. All the while, bidirectional interaction between on-site personnel and back-office operators has to be ensured.

This scenario highlights some of the complexities arising from the various interaction dependencies that subsequently need management. Merely modeling user collaboration as use cases or analyzing low-level technical design documents will not assist the design team in apprehending the implications of the implicitly emerging user interaction protocols. A design team first identifies suitable patterns, which address the underlying dependency coordination needs. There is no pattern that adequately supports all types of dependencies such as information flow, control flow, resource access, and task execution [13], but typically there are several ones that excel at a subset.[§] Investigating and evaluating the applicable collaboration patterns allow system designers to understand and subsequently answer the following example questions:

- Q1: Which pattern supports an adequate trade-off between control and flexibility when passing information from on-site personnel to back-office operators? What advantage does a publish–subscribe-based mechanism offer compared with pre-configured event routing?
- Q2: Which pattern supports information-processing constraints as well as flexible information collection before passing information between work shifts? Are shared artifacts too flexible or would workflows be too rigid?
- Q3: According to which pattern should analysts be organized to ensure timely and high-quality results? Should they work on analysis tasks jointly or rather carry out replicated tasks individually?
- Q4: Is a hierarchical structure for incidence response teams flexible enough for quick customization upon deployment?
- Q5: What impact has the deployment of emergency dispatchers (for coordinating on-site personnel and incident response teams) on flexibility?

In order to answer these questions, knowledge about the patterns' elements alone is insufficient. The system designers need a fundamental understanding of the collaboration patterns' provided adaptability: what flexible actions a particular system user role may entail and what adaptation actions the system may execute automatically. Such understanding is also of highest importance for perceiving the patterns' limitations. Overall designers need to apprehend what elements can and cannot (easily) change (*behavior*), the duration of enacting a change and its effect on the system (*asynchrony*), where collaboration awareness and know-how are captured (*state*), and under which conditions a change may be enacted (*execution*). Only then are the designers in a situation to make principled decisions and safely evaluate trade-offs among design alternatives. To this end, the BASE framework (Section 3) provides a structured approach that focuses exactly on those concerns: **B**ehavior, **A**synchrony, **S**tate, and **E**xecution.

Without in-depth knowledge on pattern adaptability, the design team might arrive at following design decisions (Figure 1): (1) implement the Log Book as a Shared Artifact (unaware of its overly flexible user involvement); (2) manage tasks in a Master/Worker manner (unsure whether more sophisticated control flow might be required); (3) enable event distribution through Publish/Subscribe (not knowing where to enforce constraints); (4) organize incident teams peer-to-peer (not considering the pattern-specific risk for long time-to-team); and (5) introduce emergency dispatchers for coordinating teams and incident reporters (unaware which participants may suffer from rigidity).

[§]A discussion on which pattern best supports a particular dependency type is outside the scope of this paper.

DOI: 10.1002/cpe

Each of these patterns has different idiosyncrasies when it comes to available collaboration-level primitives for executing adaptation actions, respectively enable user flexibility. We will discuss the detailed design decision impact and trade-offs in Section 6.

The next sections, however, introduce the BASE framework for analyzing pattern-specific adaptability, followed by an in-depth discussion of the various patterns' adaptability.

3. APPROACH

In [13], Malone and Crowston highlight the existence of similar coordination aspects in Economics, Organizational Theory, and Computer Science. 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, and secretaries are rarely described as connectors in real world environments, but they perform a similar task: the coordination of other humans (i.e., components).

As emphasized in [7], 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 these dynamic elements. Connectors, thus, are one of the key elements to system adaptability. For example, connectors allow the dynamic replacement of behavior components in robotic systems without affecting other components [14].

Collaboration Connectors enable similar capabilities among interacting users. One of the main purposes of a *Collaboration Connector* is managing the interactions among *Human Components*. In the *Master/Worker* pattern, for example, the task creator (client) and task executer (*Worker*) comprise the human components. Without the *Master* (a collaboration connector), client 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, and worker replacement. 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 necessarily implemented by a human. Typically, dedicated user roles monitor pure software-implemented connectors. Any such software, however, resides at the same conceptual level as a collaboration connector. Hence, we make no distinction between human-initiated and software-initiated actions when discussing a collaboration pattern.

The scenario demonstrates how complex, interaction-intensive systems combine multiple collaboration patterns. The system design team thus needs to specify what degree of flexibility, respectively constraints, the various user roles require in a particular pattern. The design of user-centric adaptation mechanisms (e.g., recommendations, filtering, and (semi)-automatic reconfigurations) must build on the underlying pattern's interaction primitives (e.g., collocating filtering with collaboration connectors). We thus propose to treat collaborations in terms of human components and collaboration connectors and subsequently refashion tools from the software architecture domain for evaluating a pattern's adaptability.

3.1. Behavior, asynchrony, state, and execution background

The BASE framework [7] defines four aspects relevant for software runtime adaptation: **b**ehavior, **a**synchrony, **s**tate, and **e**xecution context. Analyzing these aspects dispenses with encoding the styles in a formal language, which simplifies the transition of BASE to collaboration patterns. 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, that is, the means for changing a system's behavior and the respective level of abstraction such as reconfiguration at code level or at

DOI: 10.1002/cpe

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 maintaining system constraints during adaptation and continuous system availability.

State refers to potential adaptation 'side effects' in terms of impact on the system's state. Changing a data type definition might require updating all current instances to the new definition. Replacing 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.

3.2. Collaboration adaptability aspects

Compositions from *Human Components* and *Collaboration Connectors* produce patterns similar to architecture styles. Given the similarities highlighted previously, the BASE framework applied to collaborations considers the same runtime adaptability aspects.

Human components and collaboration connectors react to (or even anticipate) changes in the environment. The environment includes everything a human component or collaboration connector is capable of observing: the underlying collaboration, system, and its context. They thereby select actions autonomously, that is, without necessarily being told by someone else what to do but purely based on their awareness (workspace, process, interaction, context, etc.). Adaptability in collaborative settings comes in two broad forms:

- **flexibility** provided to human components (i.e., users) in terms of the available pattern-specific actions. In the *Master/Worker* pattern, for example, a client might obtain additional information rendering some open task irrelevant, which is subsequently canceled. A worker might drop poorly paid tasks in return for better paid ones;
- **control** through coordination actions available to connectors (i.e., software and/or particular users) applied for fulfilling the connector's purpose (e.g., delivering messages, assigning tasks, and controlling access) and thus for preserving a pattern's properties (e.g., publishers never interact with subscribers directly). In the *Master/Worker* pattern, for example, the master observes inconsistent task results and decides to issue additional jobs for improving the quality. Additionally, the master may no longer assign jobs to unreliable workers.

Hence, throughout this paper, the term 'adaptation' signifies both human component flexibility as well as collaboration connector-centric actions.

Behavior similarly addresses the means for adaptation. Adaptability is not limited to changing a single user's involvement. First, there are multiple levels of granularity that allow behavior adaptation. Replacement of a complete company and team, adding another worker, or acquiring a required skill all correspond to component-level adaptation in software systems. Second, behavior distinguishes between component-centric and connector-centric adaptation actions. A third behavior facet focuses on the component and connector wiring. Who is allowed, respectively empowered, to interact with whom. For example, a news consumer chooses among multiple news publishers by applying criteria such as accuracy, level of detail, or timeliness. Finally, the interaction means among users represent significant loci of flexibility and adaptation. Shared artifacts and messages might enforce a precisely specified structure (e.g., medical records and insurance claims process) while other forms provide various degrees of flexibility (e.g., an email and a Wikipedia article).

Asynchrony in collaborations refers to the time required for establishing a (new) team, replacing a worker, becoming acquainted to another worker, familiarizing with work and becoming productive, or learning a new skill. Besides changing structural pattern elements, asynchrony highlights the impact that duration and spread of revising the interaction protocol among users, updating

DOI: 10.1002/cpe

message or document formats, or reallocating resources and privileges might have. Asynchrony focuses on such immediate changes as well as the temporal side effects on the joint work during adaptation. It, hence, raises awareness on constraints that need enforcement during the reconfiguration. An example constraint might require the continuous availability of at least one former team member during a team's replacement 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 knowledgeable 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.

3.3. Human-driven adaptation idiosyncrasies

There are several aspects where software-driven adaptation and human-driven adaptation differ. The main ones being (1) human autonomy, (2) extensive awareness of one's surroundings, and (3) human cognitive capabilities.

Autonomy becomes most apparent when it comes to rewiring (introducing, removing, and changing connections among components and connectors) a collaboration instance. Software architecture-centric rewiring assumes control over all involved (i.e., rewired) elements. In collaborative settings, we cannot automatically assume such 'obedience' but must rather explicitly elaborate on which user role (component or connector) initiates a new connection, and whether and who is involved in completely establishing such a link. Variations in initiation and establishment typically represent different refinements of the same pattern. Adaptation becomes more complex as the adaptation mechanism needs to consider, for example, that recommendations are hardly followed in all cases. Yet, adaptation still operates within the boundaries of the underlying pattern and thus needs to obey the pattern's limitations. A recommendation mechanism breaks the underlying assumptions in a Workflow by suggesting a worker to interact with the preceding worker. This results in a shared state no longer managed by the workflow coordinator (human or software).

Software elements are typically constraint and fixed in their perception of their environment. Humans inevitable will exhibit different levels and types of **awareness** (workspace, social, task, and process) and changes thereof across time. Collaborators may obtain significantly more awareness than expected for their role in a collaboration pattern. Greater awareness has the benefit to enable the individual to conduct earlier or more efficient local adaptations. On the one hand, this may impede adaptation at a more global level as awareness implies tighter coupling among collaborators. A subscriber, for example, may obtain detailed context information on the information sources. The user subscriber adapts by establishing preferences reflected in her or his desired message sources. The messaging connector is subsequently unable to transparently filter and reroute messages without the subscriber noticing. On the other hand, awareness greatly improves adaptability when made available to those participants in possession of adaptation authority and/or capabilities (i.e., primarily connectors). Additionally, the awareness type should match the underlying managed dependencies. Task-sequencing dependencies, for example, call for process awareness; data input/output or resource dependencies call for workspace awareness.

The combined **human cognitive capabilities** apparent in learning, forgetting, creativity, handling uncertainty, or conflict avoidance give rise to much more complex behavior than typical software components exhibit. Hence, it is not surprising that a collaboration mechanism becomes repurposed. Typical causes are a lack of suitable coordination mechanisms or a lack of sufficient tool proficiency. This results in a mismatch between the repurposed tool/pattern (e.g., a wikipage) and the actually desired mechanism (e.g., workflow management). The participants consequently need to manually manage this mismatch and make every participant aware thereof. Automatic, software-centric

DOI: 10.1002/cpe

adaptation mechanisms are at risk to becoming counterproductive. A vandalism detection mechanism, for example, might wrongly lock a wikipage that is repurposed to capture frequently fluctuating workflow information. Patterns and BASE, however, can help to identify this mismatch and how to handle it.

At the bottom line, patterns and the BASE framework are (merely) design tools that **support** a designer in determining the best coordination means (including trade-offs) for the underlying collaboration runtime concerns.

3.4. Applying behavior, asynchrony, state, and execution during system design

The BASE framework supports the design team in various development phases and remains independent of the underlying software engineering process model (spiral, waterfall, agile, etc.). The following steps outline how a design team may apply BASE for identifying suitable collaboration patterns. These steps should not be taken as rigorous, sequential instructions but merely indicate how to potentially align BASE, collaboration patterns, and system design.

(1) Determine the various coordination needs among users.
(2) Identify patterns that address these coordination needs.
(3) Determine adaptation needs through identifying the sources and situations of uncertainty, unreliability, inconsistency, and so on.
(4) Further investigate those adaptation needs that are best managed by users (and cannot be sensibly managed at the software level).
(5) Utilize the BASE terminology for specifying adaptation requirements:

> **Behavior:** Specify who will adapt, what will change, and how often something will change.
> **Asynchrony:** Specify the constraints that must hold during adaptation.
> **State:** Specify the collaboration context shared among participants (workspace awareness, process awareness, temporal awareness, etc.), how much state they should share, and what state needs externalization before adaptation.
> **Execution:** Specify how time critical changes are and what an acceptable change delay is.

(6) Compare the adaptation requirements with each pattern's BASE properties and select best match.
(7) Optionally evaluate control/flexibility trade-offs in terms of effort required to restrict flexibility, respectively relax constraints.

The steps highlight how coordination and adaptation needs (problem domain) are tightly interwoven with collaboration patterns (solution domain). They focus exclusively on adaptability. Trade-offs among patterns will extend beyond adaptability and must consider all design criteria relevant for the underlying problem. A pattern-based perspective supports such analysis. For example, identifying all involved participants and their actions in a pattern provides a basis for further investigating acceptability issues [15] or security concerns [16].

4. APPLYING BEHAVIOR, ASYNCHRONY, STATE, AND EXECUTION TO COLLABORATION PATTERNS

Numerous collaboration patterns exist for addressing various management concerns such as coordination of (1) shared resources, (2) producer/consumer relationships, (3) simultaneity constraints, and (4) task/subtask relations [13]. In this section, we evaluate seven, real world-observed patterns that we considered in our scenario: *Shared Artifact* (4.1), *Publish/Subscribe* (4.2), *Master/Worker* (4.3), *Social Network* (4.4), *Workflow* (4.5), *Secretary/Principal* (4.6), and *Organizational Control* (4.7). We neither claim that this set is complete nor claim that it includes various refinement forms. Our main purpose is highlighting the fundamental differences among patterns by describing their structural elements and by analyzing their exhibited adaptability idiosyncrasies.

DOI: 10.1002/cpe

4.1. Shared artifact

A Shared Artifact describes any type of collaboration object where participants communicate indirectly through creation, updating, reading, and deleting of a common object. Examples are shared text documents, source code files, or a discussion board (Figure 2). This pattern achieves strong decoupling among human components as it inhibits direct interactions. Instead, all communication is limited to publicly visible manipulations of (parts of) the shared artifact.

Research in the computer-supported cooperative work domain focused early on such capabilities in the context of shared workspace systems and groupware systems [17]. Several approaches target also large-scale environments [18]. With the emergence of Wikis [19], Internet-scale collaborative editing became a universal success: the most prominent example being Wikipedia [20].

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 monitor changes, thereby enable detecting, preventing, resolving, or deciding on 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 schema update of all artifacts at once prevents all collaborators 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.

4.2. Publish/subscribe

In the *Publish/Subscribe* pattern, publishers and subscribers communicate indirectly by means of events and are not necessarily aware of each other's identity [21]. A complex connector (typically automated and message-oriented) manages event collection and distribution. Subscribers process the received events and potentially produce new events of their own.

Mailing lists (e.g., Listserv [22]) emerged among the earliest instances of Publish/Subscribe for delivering 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

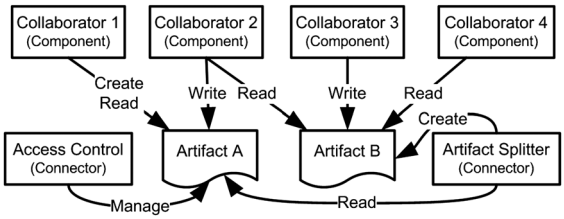

Figure 2. Schematic *Shared Artifact* pattern example.

DOI: 10.1002/cpe

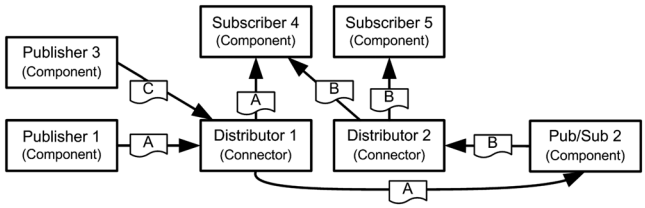

Figure 3. Schematic *Publish/Subscribe* pattern example.

reply or post their own message, and whether the list is topic or person-centric (Figure 3). In the case of mailing lists, the mailing list server obtains the role of the distribution connector.

In recent years, microblogging platforms such as Twitter have become a popular tool for rapidly disseminating information in large-scale environments [23, 24]. In contrast to purely topic-centric subscriptions in mailing lists, subscriptions in microblogging environments are mostly person-centric.

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 loose their relevance. Distributor connector replacement is only an option when published events can be buffered or rerouted. How any state (e.g., subscription status) is transferred among connectors remains outside this pattern's scope.

Distributor connector variations differ in their provided control over subscription establishment and release. A connector may automatically approve any requests or obtain a publisher's explicit consent. A connector may allow subscriptions to be initiated by a third party, different from the actual subscriber ultimately receiving the events. A connector may decline unsubscribe requests when no other subscriber remains (e.g., monitoring status of elderly people in the healthcare domain).

Event format changes typically follow two approaches: either all involved elements cease operations so that they can learn about the new format or the connector becomes format version aware and knows how to convert events on the fly. The latter approach is particularly helpful to address situations where subscribers may remain offline for a while and collect multiple events upon going online (some of which potentially may have occurred before a format update).

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, supporting mechanisms for subscribing to relevant lists is outside the pattern's scope.

In case the message distribution connector supports a single message format only, introducing a new format implies a considerable downtime as pausing all publishers, ensure that all subscribers have received pending notifications, and then updating all publishers and subscriber (i.e., notify them how to handle the new format) takes time. Publishers and subscribers would only be allowed to join again when having completed the format update to guarantee correct message interpretation.

State: The collaboration domain defines the requirements for state transfer upon swapping publishers. Distribution connectors maintain state on (1) subscriptions, (2) messages for subscribers that are allowed to go offline without missing events, and (3) publishers if the connector's purpose dictates so (e.g., monitoring whether there are sufficient publishers for a particular topic). Consumers build their internal state as new messages arrive. Most mailing list and microblogging implementations maintain a history of past messages that enable immediate state reconstruction.

Execution context: Message dispatching and receiving are considered atomic. Hence, independent unidirectional messages achieve loose coupling and limit restrictions on the replacement of publishers or subscribers. In special cases, for example, when multiple individual messages from multiple authors create a discussion thread, the removal of an involved publisher would then

DOI: 10.1002/cpe

require others to compensate. Any such coordination among publishers, however, remains out of the pattern's scope. Newly joining readers need to build the discussion thread from historical records or wait for a new discussion to emerge.

Hybrid patterns: Online discussion forums [25, 26], bulletin boards, and blogs [27] populate the spectrum between Publish/Subscribe and Shared Artifacts. Knowledge is distributed from producers to consumers, who in turn raise the message to the level of a shared artifact through refinement or extension in the form of commentary. Discussion topics and blog entries are thus more than simple broadcasted messages but at the same time do not offer the full range of manipulation capabilities that come with shared artifacts.

4.3. Master/worker

The *Master/Worker* pattern leverages parallel processing and decoupling of task clients and task executers. Partitioning tasks works particularly well in human collaboration when the resulting work items can be carried out independently (Figure 4). The client defines a work task and typically specifies the need for task replication, temporal constraints, and reward. The Master, a collaboration connector, manages the distribution of tasks in the form of jobs to available workers. Such job allocation may occur in a push or pull manner. The former procedure has the Master allocate jobs directly to workers, whereas the latter procedure enables workers to choose which job they prefer to work on. Depending on the Master's capabilities and task type, it will reallocate overdue jobs from unresponsive workers and aggregate the results from replicated jobs before returning them as task outcome to the client. A most basic Master merely collects the results and relays them back to the client who then has to aggregate the individual results itself. The large-scale deployment of the Master/Worker pattern is often referred to as *crowdsourcing* [28]: the most prominent example being Amazon Mechanical Turk [29].

Behavior: The client decides upon the number of workers that may work in parallel on copies of the same task artifact. Alternatively, it specifies quality and temporal constraints, and a sophisticated Master determines when and how much to replicate jobs. Clients going offline have no impact on other clients but need to indicate whether the Master should revoke claimed jobs, retain results for later collection, or redirect results to a replacement client.

In the pull-style assignment, workers choose which tasks to perform and whether to return a task unfinished. In this case, the Master may remain completely unaware on the number and skills of available workers, merely maintaining state on open and claimed jobs. In push-style assignment, workers receive new tasks in their job queue but may still have the option to reject or delegate a task. Here, the Master immediately notices leaving workers thus gaining the ability to reason on the impact for completing pending task within time and quality. A Master may thus better manage workers with scarce skills.

Changing a task's data format affects only new task instances. Claimed jobs typically require cancelation and reposting to reflect any updates. On the one hand, when format updates affect only the task's description, the Master remains unaffected, and only workers attempting to claim

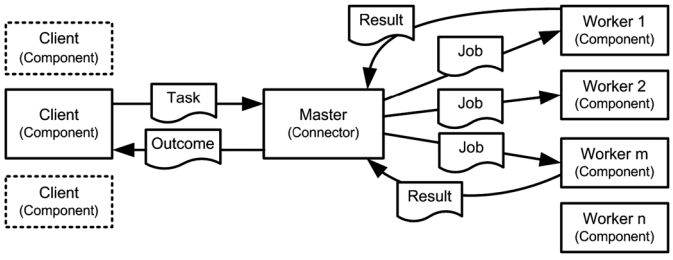

Figure 4. Schematic *Master/Worker* pattern example.

DOI: 10.1002/cpe

such a job require updating. On the other hand, changes that affect the Master in managing task (e.g., allocation, replication, and aggregation) require Master replacement or reconfiguration (and updates of other clients when only a single format is supported) but remain invisible to workers.

Asynchrony: A task artifact completely decouples clients and workers. The client, respectively the Master (assignment connector), has the option to reassign the task to another worker (respectively make it available again) when the worker fails to complete the task in a predefined time frame. The client informs the Master on the number of multiple identical tasks for the sake of reliability or the issuing of multiple sequential tasks until the desired result quality has been achieved. Thus, the impact of unreliable workers on quality, cost, and timing highly depends on the chosen allocation strategy.

Task format updates have little temporal impact as long as they remain specific to the task descriptions. Hence, only the workers need to learn how to handle the differently specified job. Format updates to task management related information have a similar impact as event updates in the Publish/Subscribe pattern. In case of a single valid task format, all clients have to update for future task request, and the Master had to stop and learn how to handle the new format. Otherwise, the Master requires version capabilities but nevertheless will be unavailable during updating.

State: The task artifact contains the complete collaboration state. Replacement of workers has no side effect on the state. The Master, however, maintains state on which jobs are yet unassigned, which worker claimed particular jobs and their respective due dates. Master replacement thus requires the ability for externalizing and handing over such state information.

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 (i.e., jobs) and have no knowledge about each other. They remain similarly unaware of any replacement of the client. A new worker simply obtains the task description and commences task execution independently of any previous work done. Replacement or updating of the Master is non-trivial as this involves transition to a state of quiescence, buffering, or redirecting incoming requests and results, as well as externalizing state on tasks, jobs, and workers.

4.4. Social network

Social Networks (e.g., Facebook and LinkedIn) are characterized by a large number of participants, sparse connections among them (i.e., the number of acquaintances is small comparative to the overall size of the network), and a lack of centralized control (Figure 5). A social network's topology typically exhibits multiple communication paths between any two participants, some of these of very low hop count [30]. Power-law distribution of links is found throughout most social networks: a few members are extremely well connected, while the majority exhibits only few links [31]. A social network's two main purposes can be best described as keeping acquaintances informed about oneself and processing 'service' requests of any kind (e.g., who knows a good restaurant and who could help me with . . .). Processing may include issuing of, filtering of, forwarding of, and replying to requests.

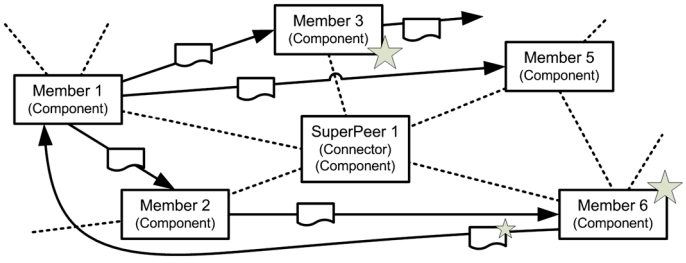

Figure 5. Schematic *Social Network* pattern example.

DOI: 10.1002/cpe

Social networks provide an adequate substrate from which self-organizing teams without an a priori specified coordination structure emerge. Instead, individual members collaborate in an ad hoc fashion to complete a common goal. The flexibility provided by social networks allows individual members to become more active whenever they see fit without incurring much coordination overhead. Newcomers receive briefings from multiple members and thus are quickly brought up-to-date. Multiple communication paths maintain the information flow in case a member becomes temporarily or indefinitely unavailable. Many open-source software development efforts build upon social networks of developers, which allow them to form self-organizing teams [32]. Self-organization, however, also has its downside as the effect of (flash) mobs demonstrates. Another challenge poses the mobilization and continuous engagement of sufficiently many participants to keep the collaborative momentum alive.

Behavior: Regardless whether social network or self-organizing team, every *Member* is free to leave abruptly, and new members join spontaneously. A participant remains flexible in deciding with whom to establish links (unidirectional link) or who to accept as acquaintance (bidirectional link) and how long to maintain a link. They thus adapt internally when (1) updating their network view to reflect, for example, unresponsive or exploiting members, as well as (2) on demand when deciding when to process, who to forward to, and whether to fulfill a particular request.

Without centralized control and self-managing members, no external entity can directly 'adapt' individual peers. Where available, the underlying network management platform (e.g., a social network platform) might analyze user profiles, behavior, and interactions for proposing recommendations on who to interact with or what member updates to display. Each member, however, remains in complete control over who to contact, who to obtain information from, or who to maintain in their 'friend list'.

Collaborative efforts in a social network that require increased levels of coordination (e.g., open-source software development) typically experience the emergence of *Super Peers*. These are members who have a special interest in the effort, have proven stable within the network, might possess special resources, and respectively are willing to provide them (e.g., skills or time). Again, such super peers are not a priori defined but take on these coordination duties on their own account. Super peers thus exhibit collocated connector and component functionality.

Asynchrony: A social network is never stable. There is always a subset of fluctuating members at any single point in time. These fluctuations, however, hardly disrupt the underlying social structure in the presence of sufficiently many members and links. Retirement of super peers potentially degrades a collaborative effort's performance until another member assumes the vacant position.

Changes such as updating a participant's acquaintance list or who will be considered for requests remain local, limited to the involved participants and thus never require waiting for the network to stabilize. For the same reason, message specification updates may not occur simultaneously across the whole network. One of the main benefits of having a single network management platform lies in the percolation speed when conducting such updates. Members have to 'learn' the new format to continue participating. In distributed platforms, members might decline and cause a network partition along the old and new message format.

State: Collaboration state is generally spread about multiple members (i.e., those that were involved in joint activities). Mechanisms for externalizing member state such as shared artifacts or message history are outside the scope of the pattern. New members gather the required state information from existing peers, explicitly by querying them and also implicitly when processing and forwarding new incoming requests. Super peers assist in rapidly establishing the collaboration state by recommending relevant existing members to connect to.

Execution context: Members engage in joint activities and are free to leave at any time. In work intensive efforts such as open-source development, however, they are expected to either complete their activities before leaving or at least externalize the task-relevant part of their internal state. Abruptly leaving members may result in lost collaboration know-how. In the ideal case, work is subsequently picked up by other network members, simultaneously communicating with their neighbors and super peer for establishing sufficient state before commencing work.

DOI: 10.1002/cpe

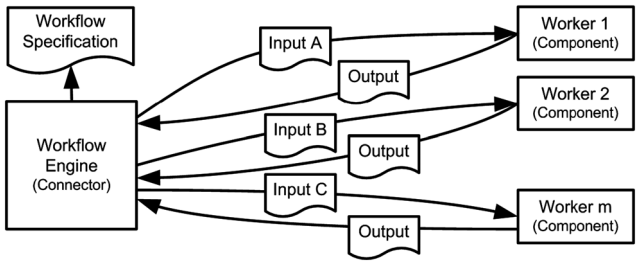

Figure 6. Schematic *Workflow* pattern example. Temporal and causal relations among workflow tasks are internal to the workflow specification and thus not shown.

4.5. Workflow

In a sequential *Workflow* such as an assembly line, a worker picks up a work item in his or her queue (e.g., a document and a product part) and manipulates it according to the work description. He then forwards the work item to the next position in the flow before continuing with the next available item in his work queue. Interaction between individual workers is mostly determined through the shared work artifact. Direct communication between workers is limited (if possible at all) to coordination of load differences and delays between neighboring work stations.

With the emergence of office automation systems [33], nonlinear workflows became prevalent. Evolved into process management systems, automated coordination of human work [13] resulted in individual workers becoming increasingly unaware of their preceding or successive work steps. A central workflow engine dispatches tasks to the various workers, collects their output, and decides where to route the artifacts next (Figure 6). The focus on the technical coordination aspects [34] over the last decades resulted in a rigid decoupling of individual workers. Workflow description languages dedicated to modeling the human involvement such as Little-JIL [35] or BPEL4People [36] foresee no explicit communication between workers. We apply BASE only to the nonlinear workflow pattern, rather than assembly lines, as they are prevalent in modern interaction-intensive systems.

Behavior: On the one hand, adaptation occurs at the level of the workflow structure (rewiring of workflow steps) where workers would execute the same task with the same input but in a different sequence. On the other hand, workers may also be removed from one task and added to another one. In addition, switching or updating a task's artifact specification reflects a change in input/output data. This may entail (1) having to engage a worker with different skills and/or (2) reflecting the changes in the workflow's specification.

Any adaptations to the workflow engine (a connector) itself require pausing or completing all running workflow instances. New instances have to be buffered or redirected to another available engine.

Asynchrony: In artifact-centric assembly lines, replacement of workers causes work items to remain longer in the queue as workers switch places. More complex rewiring of work stations requires feedback to previous workstations to limit their production rate. In information-centric workflow systems, replacement of a worker is equivalent to moving all unprocessed tasks from the worker's in queue to the new worker's in queue and waiting for the leaving worker to complete his current task assignment. There is no coordination required between any human workers.

Restructuring of the overall workflow structure is usually limited to new workflow instances. Adaptation of ongoing workflow instances requires the affected workflow parts to reach a stable state (e.g., no active task executions). The same goes for updating task artifacts.

DOI: 10.1002/cpe

State: In assembly lines, all state is represented by the progress of the work artifacts. In non-linear workflows, state is maintained by the workflow engine. Thus, in both cases individual workers remain stateless with respect to the overall process and can be replaced between two task instances. As the workflow specification does not foresee explicit communication among workers, any occurring implicit communication may capture collaboration state. Such state is prone to loss upon worker replacement as no formal mechanisms are in place for externalization and handover.

Execution context: Coordination between adjacent workers in an assembly line requires task completion before the actual worker replacement may be carried out. In workflow systems, no such communication is foreseen. In both cases, a worker needs to complete his current assignment before any adaptation can occur. Mechanisms for worker replacement during long-running task executions are external to this collaboration pattern. Depending on the workflow engine's capabilities, any update to the workflow specification is limited to future instances and thus can be immediately enforced for the upcoming instance, or also for currently running instances. In the latter case, the engine needs to bring the workflow to a stable state and subsequently ensure that the output of all already completed activities is compatible with the updated specification. Achieving a stable state is equally important when a workflow engine's adaptation has to occur during a long-running workflow and cannot wait for its completion. New instances have to either wait or commence on another workflow engine instance.

4.6. Secretary/principal

In human collaboration, secretaries (or assistants) introduce layering to collaboration. Clients cannot contact the desired person directly. Instead, they first pass their request for a particular 'service' to the secretary who acts as an intermediary between client and principal (Figure 7). In other words, clients typically require a particular capability or functionality (e.g., authorization, information, and calculation) from a principal but do not necessarily require it from a particular principal instance. A secretary forwards the message to the principal and relays the response back to the client. Alternatively, a secretary may respond immediately on behalf of the principal. Depending on the particular collaborative setting, secretaries serve as load-balancing proxies, protection proxies, caching proxies, or brokers [37].

Behavior: Clients are volatile and expected to contact a secretary unannounced. Secretaries and principals may be replaced anytime; however, their relation is longer lasting than between client and secretary. The set of supported message types and artifact types varies for each secretary and principal instance. Associated secretaries and principals, however, necessarily support overlapping type sets. A single secretary may be coupled to more than one principal and vice versa.

Secretaries may be easily replicated as they basically forward requests to the principal. This allows the client to contact a random secretary (provided they handle the same requests) and thereby remain unaffected by a particular secretary's availability (e.g., vacation).

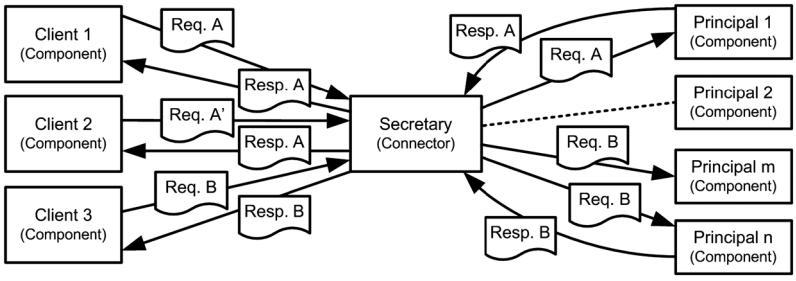

Figure 7. Schematic *Secretary/Principal* pattern example.
BETTER: with example requests (Req.) and responses (Resp.).

DOI: 10.1002/cpe

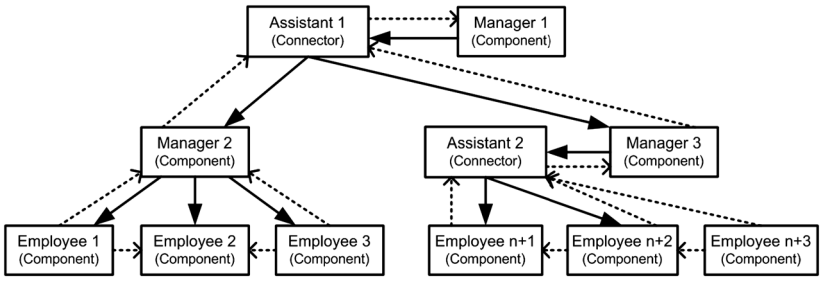

Figure 8. Schematic *Organizational Control* pattern example.

Asynchrony: The secretary may decide on how to respond and on how to route the request depending on the client's request and the principal's availability. In case neither the principal nor the secretary is able to respond, the secretary can ask the client to repeat the request at a later time. Where supported, the secretary may offer to put the request on hold and notify the client once the principal becomes available again.

State: Secretaries are initially stateless and build up any required internal state from interactions or querying the principals for preferences. Having the new secretary temporarily observe the current secretary enables rapid establishing of state. Alternatively, a secretary can refer clients to another secretary during replacement.

Execution context: A principal may decide to refuse responding to any requests, forcing the secretary to reply on behalf or reroute the request to another principal. A secretary needs to complete all current requests before replacement but can redirect new requests to other assistants.

4.7. Organizational control

Hierarchical organizations are found in diverse domains: multinational corporations, military, government, and universities exhibit pyramid-like levels of management. Organization control comes in two broad forms: behavior control and output control [38]. The supervisor applies (pro-active) behavior control (i.e., downward requests) to trigger a specific, desired behavior. Output control describes the manager monitoring work progress (i.e., upward events) to reactively maintain, or respectively restore, work performance (Figure 8).

In strict hierarchies, a manager issues a request to one of her subordinates on a layer below her. The subordinate in turn responds to the request and/or issues his own requests to employees on layers below him. Employees on any layer report on work progress or problems via events to their supervisors on levels above them. With many layers and strict communication paths, information remains tightly controlled but also requires considerable time to spread throughout the organization [39]. Likewise, ad hoc collaboration among leaf elements in different hierarchy branches is closely impossible or highly discouraged (e.g., especially in military settings).

While requests and notifications are typically sent directly to specific employees, messages are bound to roles and functions. Roles imply particular capabilities and thereby enable decoupling of request addressing from the actual employee executing the request.

Behavior: Organizational charts describe the links between the different roles and the mapping to actual humans but cannot provide request and event buffering. Adaptation corresponds to structural updates of the organization chart in terms of reassigning employees to roles and rewiring role relations. As such, individual employees have little flexibility: they are assigned to a particular role and must obey specific interaction paths. Replacement and relocation of employees (i.e., as dictated by organization chart updates) remain centralized at a dedicated department (e.g., Human Resource) that per se is external to the organizational structure's function and purpose.

DOI: 10.1002/cpe

Assistants are one form of connectors but are usually not universally implemented (i.e., typically only in management). Organization control exhibits similarities to the *Secretary/ Principal* pattern. The intermediaries, however, are most often applied to management levels only. The major difference lies in explicit, request-independent upward notifications, which are absent in the Secretary/Principal pattern.

Asynchrony: When replacing an employee or rewiring a role, lower-level subordinates need to buffer notifications until adaptation has completed. Likewise, up-level supervisors will not receive notifications for the adaptation duration and have to refrain from dispatching requests to the affected employee. With assistants or secretaries in place, responsibility for buffering of requests and notifications is transferred from employees to those connectors.

State: Individual employees maintain their own state. State transfer is only required between two switching employees when collaboration know-how remains employee internal and cannot otherwise be made explicit. New employees can apply the organizational chart to contact colleagues to retrieve (additional) information necessary for constructing state. Organizations typically store general purpose state information in corporate 'data warehouses' at the lowest relevant level in the hierarchy. Accounting data, for example, remains high-up at the management level, whereas travel guidelines are placed at the bottom of the hierarchy available to every employee.

Execution context: Any ongoing request needs to be completed before an employee can be replaced. If an employee becomes unavailable during request execution, the request has to be addressed again with her substitute employee.

5. USE CASE EVALUATION

Having analyzed the seven patterns' adaptability properties, we now return to the design concerns we raised in the motivating scenario. For each question, we discuss how BASE assists in elaborating the adaptation needs and subsequently outline how the various applicable patterns support, respectively neglect, those requirements.

Q1: *Which pattern supports an adequate trade-off between control and flexibility when passing information from on-site personnel to back-office operators?*

On-site personnel needs the freedom to flexibly decide on what information to spread. Likewise, back-office operators flexibly need to select information sources. In addition, the selected information-passing mechanism needs to enable dynamically changing participants (**Behavior**). Any such changes (available participants and information flow) need to occur instantaneously (**Execution**) without affecting non-involved participants (**Asynchrony**).

Two patterns emerge as the most suitable candidates for managing information flow dependencies: *Publish/Subscribe* and *Workflow*. *Publish/Subscribe* enables back-office operators to pull the information they require, dynamically adapting their set of required information sources and dropping sources that seem irrelevant. In contrast, sequential, *Workflow*-controlled message routing pushes events to particular user roles. This pattern allows tight control over the types and sequences of workers processing the events as all communication paths are predefined. Simultaneously, it severely restricts operators in selecting information sources, as well as what information they intend publishing themselves. Adaptation control thus lies with the workflow connector, leaving little user flexibility.

With Publish/Subscribe, a major concern arises: given too much flexibility, certain parts of the critical infrastructure might end up unattended. Adding constraints to *Publish/Subscribe*, however, is simpler than increasing user flexibility for specifying input/output of individual *Workflow* steps. The system designers thus decide on enhancing the publish/subscribe message distribution connector with default subscriptions linked to particular subscription roles. A sophisticated connector is role and subscription-aware; thus, it can adapt for ensuring that all areas remain monitored (e.g., decline unsubscribe requests). All the while, publishers and subscribers remain strongly decoupled and maintain their flexibility.

DOI: 10.1002/cpe

Q2: *Which pattern supports information-processing constraints as well as flexible information collection before passing information between work shifts?*

Before the collected information is handed over to the subsequent work shifts (**State**), it needs aggregation, brief verification, and finally also anonymization. The design team needs to balance this requirement with a need for back-office operator flexibility during information collection (**Behavior**). This balance implies that changes cannot be enforced at anytime (**Execution**).

A *Shared Artifact* seems a flexible, robust choice for aggregating and passing information between work shifts. It relieves members of the earlier shift of directly knowing which operators will be accessing the information next. In return, operators in a later shift need not request such log information from their predecessors directly. An access connector monitoring manipulations to the log book would observe changes but would neither be able to distinguish among different forms of manipulations (aggregation, verification, and anonymization) nor roles entrusted with those manipulations nor be able to enforce temporal ordering of these steps. The *Shared Artifact* pattern does not foresee such constraints and subsequently exhibits too much flexibility.

The *Workflow* pattern presents a suitable alternative when centered around artifacts [40]. Workflow systems enable flexibility [41] through partially relaxing the precise sequence and cardinality of certain steps (e.g., allow multiple content aggregation and manipulation as long as no validation has occurred) while controlling input and output of individual steps and their associated user roles. Note the difference in flexibility compared with the workflow aspect in Q1: here, (in Q2) the task sequence remains flexible while the data specification is rigid. In the previous case, the steps remain fixed, but the input and output specifications required flexibility.

The design team balances worker step selection flexibility with control over the artifact status. A flexible workflow engine enables user task selections to be immediately applicable and restricts adaptation effects to the particular, underlying workflow instance. The design team is thus able to appropriately select the Workflow pattern over the initially chosen Shared Artifact pattern.

Q3: *According to which pattern should analysts be organized to ensure timely and high-quality results?*

Monitoring of critical infrastructure requires the ability to rapidly process a plethora of events when safety critical situations arise. Several patterns coordinate such resource utilization dependencies (here back-office analysts working on tasks). The primary adaptation concerns are, therefore, high flexibility in involving specific analysts (**Behavior**), high flexibility in deciding on the next task (**Execution**), rapid formation, assignment, and cancelation (**Asynchrony**), as well as externalizing task state (**State**).

Workflow specifications remain too be rigid for ad hoc adaptation as it might not be clear from the triggering event what subsequent process to instantiate. *Organizational control* provides a clear communication structure but lacks information-processing speed and scalability (typically each function/role is available only in limited quantities). An enterprise *Social Network* provides the opportunity for establishing ad hoc teams and thus solving complex analysis problems, but the formation process requires considerable time.

Eventually, the design team settles on the *Master/Worker* pattern. Simple tasks can be rapidly assigned to workers and replicated where required for quality assurance. Task independence further allows the straightforward analyst involvement across distributed locations as no coordination among analysts is foreseen. Simplifying and standardizing analysis tasks further allow sharing the analyst pool among multiple monitoring stations, further reducing costs and increasing response time. The Master maintains control over task execution by replicating and reassigning tasks. The back-office operators (task clients) remain flexible by dynamically chaining different analysis tasks from analyst with the appropriate skills without having to interact with them directly.

Q4: *Is a hierarchical structure for incidence response teams flexible enough for quick customization upon deployment?*

Incidence response teams have to deal with varying situations that cannot be foreseen upfront and which are subject to rapid changes as the situation unfolds. Fundamentally, this corresponds to identifying the appropriate resources and subsequent task processing by those resources. Adaptation needs arise from the uncertain resource available (**Behavior**) and inability

DOI: 10.1002/cpe

to a priori specify optimal communication paths; that is, topology changes need to be without delay (**Execution**), independent of other participants (**Asynchrony**), and enforceable by all participants (**Behavior**).

Organizational control established precise roles and communication paths but expects centralized, top-down control for reconfigurations. Given the precise specification of roles and their responsibilities, this pattern provides high flexibility when selecting appropriate employees fulfilling those roles but less so for ad hoc adding or removing capabilities.

Taking inspiration from *Social Networks* and self-organizing teams, the inverse approach to structuring the incidence response team consists of having the team members establish their links dynamically as the underlying situation requires. Such a team can identify missing skills more rapidly and keeps communication paths extremely short. On the downside, self-organizing teams beyond a certain size cannot guarantee success in safety critical environments.

The design team identifies this threat and subsequently decides upon a team coordination infrastructure with a small core, hierarchically organized member set, which is empowered to dynamically add further members as the situation dictates. Those auxiliary members then communicate among each other directly while the core team behaves as a composed super peer. This hybrid structure combines individual member flexibility with controlled adaptation due to the guaranteed availability of a super peer connector.

Q5: *What impact has the deployment of emergency dispatchers (for coordinating on-site personnel and incident response teams) on flexibility?*

Customer requirements force the design team to introduce emergency dispatch personnel for relaying requests from on-site workers and back-office operators to incident response teams. Applying the *Secretary/Principal* pattern comes with a number of advantages such as the ability for dispatching the closest or most suitable team, keeping other teams informed, and providing a unified access point for inquiries (**State**), thus reducing the number of request having to be forwarded to the actual team.

The design team, nevertheless, needs to understand the implications by having this layer of indirection. Emergency dispatchers might cause, for example, needless delays when passing on critical information from on-site workers to members of the incident response team. Introducing a mechanism for on-site workers to join the incident team in an ad hoc fashion mitigates this shortcoming (**Behavior**).

6. DISCUSSION

The seven discussed collaboration patterns exhibit significant adaptability capabilities. Just like their architectural styles counterpart, they achieve desirable adaptability properties only under particular conditions. Choosing one pattern over another for a system design, hence, is not a simple matter of comparing high-level pattern aspects. First, patterns address different dependency management issues such as control flow versus information flow. There is little point in comparing *Publish/Subscribe* with *Secretary/Principal* when the main dependency concern is information flow. This also implies that patterns and BASE cannot be sensibly applied independently but only in combination. Second, each system design has a different adaptation focus and constraints: whether adaptation control should reside with autonomous participants (e.g., collaborators on a *Shared Artifact*) or primarily with dependency coordinators (e.g., the master in *Master/Worker*). Third, software support significantly determines the need and scope of adaptation. A workflow connector implemented entirely in software (i.e., a workflow engine) implies typically more complexity and less adaptability than a human coordinator overseeing and intervening in the workflow execution. We therefore refrain from assigning qualitative ratings from low to high for each pattern and individual BASE facet in Table I. We rather discuss pattern traits that give rise to adaptability.

One of the patterns' common properties promoting adaptability is strong collaborator decoupling. The authors of the original BASE framework [7] point out three adaptability-fostering mechanisms in the context of architecture-driven adaptable software systems. They highlight (1) identifying the

DOI: 10.1002/cpe

Table I. Overview of Behavior, Asynchrony, State, and Execution properties for each collaboration pattern.

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

DOI: 10.1002/cpe

exchangeable parts and rendering them malleable, (2) managing the interactions involving those parts, and (3) explicit state management. The same strategies apply well to human collaborations. Design decisions are thus concerned with

- identifying collaborator roles (in the scope of a particular collaboration pattern), determining their required level of flexibility, specifying conditions when to restrict such flexibility (i.e., adaptation actions), and how to enact adaptation actions (e.g., recommendations versus various degrees of automatic intervention);
- selecting a collaboration pattern that supports interaction management at the desired level of control. A *Workflow*, for example, provides tighter control over user involvement than a *Shared Artifact*. Within the selected pattern, a collaboration connector assumes the actual interaction coordination (e.g., routing, buffering, filtering, and preprocessing) and hence becomes the logical place to place adaptation effectors. Along these lines, the pattern determines applicable connector capabilities; and
- specifying what collaboration state needs to be made explicit and what pattern supports this information most naturally (e.g., a shared artifact, workflow progress, subscription information, and task results).

All patterns (except *Social Networks*) exhibit loose coupling of collaborators. This enables adaptation mechanisms to limit the impact of reconfiguration actions (e.g., member replacements, rewiring, and artifact recommendations) to a subset of all collaborating users. In the scope of our motivating scenario, the patterns enable following example adaptations: recommending undersubscribed information sources without the publishers having to worry about how many subscribers exist (*Publish/Subscribe*), substituting an operator taking care of log book data anonymization without other operators noticing (*Workflows*), back-office operators (the task clients) adjusting their work shift cycle without affecting pending analysts' tasks (*Master/Worker*), dynamically introducing additional dispatchers for load balancing without having to inform incident response teams (*Secretary/Principal*), and replacing an analyst in the management hierarchy (*Organizational Control*).

Social Networks, in contrast, emerge from direct connections among participants by definition. In the general case, this leads to tight coupling as participants communicate directly (i.e., without any intermediaries such as connectors). Participants, however, expect their neighbors to be (temporarily) unresponsive or be unable to process a request. Social networks thus embrace dynamics and high failure likelihood (per request) and mitigate those shortcomings through promoting a combination of request replication, multiple communication paths, and ad hoc introduction of super peer connectors.

It can be argued that decoupling users leads to a reduction in communication bandwidth and subsequently renders the collaboration less efficient. Direct communication among participants, however, is no longer feasible beyond a certain collaboration size. Decoupling becomes a necessity as it enables specification of roles, responsibilities, communication paths, and dedicated coordination elements, which in turn increases efficiency.

6.1. Limitations

Patterns per se cannot guarantee a collaboration structure's adaptability. Human factors such as social ties, previous interactions, and trust have considerable influence on flexible users' actions as well as determining whether any connector-centric adaptation mechanisms will be effective. Independent of these properties, however, the underlying pattern provides a set of constraints that determines the scope of adaptation actions. A pattern determines the relevance of particular human factors. The *Master/Worker* pattern, for example, emphasizes worker skills and reliability rather than trust among workers. In contrast, subscribers will trust information from known publishers more than information from newcomers.

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 user to add, and what artifacts to recommend). We, however, must stress that such factors cannot be properly exploited without understanding the pattern's implications. After all, adaptation

DOI: 10.1002/cpe

actions are situated within a particular (composite) pattern and thus must work within the limits of that pattern. Experiments will give insight into these questions. Evaluations of real world systems also serve to demonstrate our contributions validity beyond analogies with software engineering. To this end, we applied BASE to Wikipedia, Twitter, and Amazon Mechanical Turk to demonstrate how adaptability of the generic *Shared Artifact*, *Publish/Subscribe*, and *Master/Worker* patterns, respectively, may be realized in successful, large-scale systems [10]. We additionally investigated how to achieve pattern-based adaptation of Wikimedia [9].

6.2. Engineering synergies among software-intensive systems and collaborative systems

Ideas, concepts, and mechanisms from the software architecture domain have inspired and guided our contribution throughout most of this paper. The distinctions between software components and connectors reflect in human components and collaboration connectors. The BASE framework informs us on key aspects that determine adaptability. The benefits of architectural styles motivate the classification of collaboration structures into patterns. We can subsequently compare patterns in terms of their admissible user-centric flexibility, respectively the remaining system control: from mostly self-organizing *Social Networks*, *Shared Artifacts*, and *Publish/Subscribe*, to layered *Secretary/Principal* and *Master/Worker*, to tightly managed *Workflow* and *Organizational Control*. The success of Software Architecture Description Languages has encouraged us to investigate a corresponding modeling approach for collaboration topologies. The interested reader finds details on our human Architecture Description Language in [9].

Luckily, the impact of software architecture research needs not remain one-way only. We strongly believe that investigating adaptability in collaborative systems can in return influence research on architecture-centric software adaptation. Ultra-large-scale systems [2], for example, defy central and hierarchical adaptation managers. In this paper, we highlighted the fundamental difference between system-initiated control (i.e., enacted through collaboration connectors) and the flexibility demanded by independent, autonomous user (i.e., human components). It seems reasonable to expect that adaptation mechanisms addressing human collaborations are suitable candidates for designing new techniques for managing software system comprising massive, autonomous, and unreliable components. Individual, self-governing components cannot be directly manipulated but accept only recommendations. They might follow the recommendations or choose to ignore them based on internal, unobservable constraints. Subsequently, human-inspired properties such as trust, reputation, and cooperativity potentially become applicable to software components. Software entities might take context into account when deciding upon cooperation. In turn, uncooperative components may face resource restrictions or have access limited to noncritical resources.

7. RELATED WORK

Related research efforts are scattered across multiple scientific fields. Among the few interdisciplinary works, Malone and Crowston highlight the existence of similar coordination aspects in Economics, Organizational Theory, and Computer Science [13].

Software engineering efforts that specifically focus on social or collaborative aspects in large-scale systems are still rare. Existing work mainly addresses the general idiosyncrasies of Web 2.0 but remains unaware of specific interaction structures at runtime [42]. Model-driven Web engineering approaches so far focus primarily on software aspects [43, 44] and do not go beyond (user) context-centric adaptations [45, 46]. Gregg [47] discusses vital aspects to enable collective intelligence but does not elaborate beyond general design guidelines. Requirements elicitation and specification approaches consider collaboration (e.g., Collaborative Systems Requirements Modeling Language [48]) or adaptation (e.g., [49]) but omit the effects of patterns on adaptation flexibility.

Tamburri *et al.*[50] have investigated the nature of organization social structure of global software development teams. They uncovered a set of 13 community types ranging from social networks to Project teams, Work groups, Problem-solving Communities, and Communities of Practice [51].

DOI: 10.1002/cpe

They specified a decision tree based on criteria such as Situatedness, Informality, Cohesion, and Governance for classifying a given collaborative effort [52]. As such, their classification focuses mostly on the purpose and social relations among members rather than their underlying collaboration patterns. We believe that jointly investigating those communities in more detail will highlight suitable collaboration patterns that best support a particular community type.

In the domain of collective intelligence, [53] and [54] discuss general aspects of large-scale collaboration systems but do not address the corresponding adaptation flexibility. Malone, Laubacher, and Dellarocas [54] 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 [53] analyze whether collaboration is explicit or implicit, relies on existing data and 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.

Most research, however, 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 among interconnected experts [55]. Kittur *et al*. outline how the Map-Reduce architectural style enables crowdsourcing of complex workflows [56]. Dustdar and Gaedke propose context-dependent selection of collaboration structures for task delegation [57]. Several research efforts focus on the inverse and aim to extract patterns from actual interactions. Dustdar and Hoffman provide mining algorithms for collaboration patterns in enterprise settings [37]. Schall *et al*. propose the Broker Query Description Language to detect experts that bridge separate communities [58]. Analysis of message flows in twitter exposes properties of large-scale social Publish/Subscribe systems [23]. Observations of twitter messages also show how this pattern is applicable to social networking purposes as well as information dissemination [59]. Team automata aim to formalize the interactions among multiple participants in groupware systems [60]. They initially targeted Computer Supported Cooperative Work systems to rigorously define and enforce collaboration protocols [61]. Their nature, however, lends them more to the analysis and design of access security mechanisms rather than reasoning about collaboration adaptivity [16].

These research efforts remain orthogonal to our work as they provide more specific adaptation strategies and mechanisms. Our adaptability framework and analysis provide insight into the applicability of such work when analyzing these approaches in the scope of their respective collaboration pattern.

8. CONCLUSIONS

Designing adaptive, interaction-intensive systems is far from trivial. Software engineers need to identify appropriate interaction structures and evaluate their inherent user flexibility as well as potential mechanisms for controlling interactions. They need to conduct trade-offs among pattern alternatives and determine the right pattern refinements. Our approach assists the engineers in this design process by providing a unified approach for evaluating a pattern's adaptability. To this end, we demonstrated how concepts and mechanisms from the software architecture domain may be applied. Specifically, the BASE framework evaluates a collaboration pattern's adaptability by describing how behavior, asynchrony, state, and execution context either promote or inhibit adaptation. Our application of BASE to seven patterns exemplified the adaptation diversity in coordination mechanisms.

Investigating collaboration patterns is only a first step toward design support for adaptive, interaction-intensive systems. There are several open challenges we have only started to address, and many more remain open. First, engineers need a modeling language in which to specify interaction structures. We have devised a basic version of the human Architecture Description Language [9], but further refinements for specifying adaptation authority or flexibility conditions require considerable additional research efforts. Given that the presented abstractions reside at the architectural level, investigations in this direction would also further explore the applicability of lower-level design mechanisms such as polymorphism or object-oriented design patterns. Currently,

DOI: 10.1002/cpe

it is unclear whether the applied analogy would remain valid at a more fine-grained level. Second, design activities need rigorous guidelines under what conditions certain patterns can be composed and what side effects this may incur. For example, the *Master/Worker* pattern typically relies on replicating tasks for quality assurance and thus aims to restrict communication among workers. Hence, recruiting workers from a *Social Network* might incur the risk of violating this assumption. Third, a composed pattern needs to undergo analysis for providing assurances to the software engineer that his or her design will indeed behave within given boundaries as expected. We envision modifying or extending existing formal specification languages and algorithms for determining the pattern's properties such as resource usage, failure likelihood, end-to-end latency, or absence of bottlenecks.

ACKNOWLEDGEMENTS

This work is supported in part by the Austrian Science Fund (FWF): J3068-N23. The authors would like to thank the reviewers for their valuable, insightful comments that greatly helped to improve this paper.

REFERENCES

1. BETTER SOURCE: O'Reilly Tim. What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & Strategies*: 65:17. First Quarter 2007. Available at SSRN: http://ssrn.com/abstract=1008839 [Accessed on 17 November 2015].
2. Northrop L, Feiler P, Gabriel RP, Goodenough J, Linger R, Longstaff T, Kazman R, Klein M, Schmidt D, Sullivan K, Wallnau K. Ultra-large-scale systems - The software challenge of the future. *Technical Report*, Software Engineering Institute: Carnegie Mellon, 2006. Available from: http://www.sei.cmu.edu/library/assets/ULS_Book20062.pdf [Accessed on 17 November 2014].
3. Kephart J, Chess D. The vision of autonomic computing. *Computer* 2003; **36**(1):41–50. DOI: 10.1109/MC.2003.1160055.
4. Cheng B, de Lemos R, Giese H, Inverardi P, Magee J. Software engineering for self-adaptive systems: a research roadmap. In *Software Engineering for Self-Adaptive Systems, Lecture Notes in Computer Science*, Vol. 5525, Cheng B, Giese H, Inverardi P, Magee J (eds). Springer Berlin / Heidelberg, 2009; 1–26. DOI: 10.1007/978-3-642-02161-9_1.
5. Oreizy P, Gorlick MM, Taylor RN, Heimbigner D, Johnson G, Medvidovic N, Quilici A, Rosenblum DS, Wolf AL. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems* 1999; **14**:54–62. DOI: 10.1109/5254.769885.
6. Kramer J, Magee J. Self-managed systems: an architectural challenge. *International Conference on Software Engineering*, Minneapolis, MN, 2007; 259–268. DOI: 10.1145/1253532.1254723.
7. Taylor RN, Medvidovic N, Oreizy P. Architectural styles for runtime software adaptation. *WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on Software Architecture, 2009 & European Conference on Software Architecture 2009*, Cambridge, UK, Sept. 2009; 171–180, 14–17.
8. Taylor RN, Medvidovic N, Dashofy EM. *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing: Hoboken, NJ, 2009.
9. Dorn C, Taylor RN. Architecture-driven modeling of adaptive collaboration structures in large-scale social web applications. *International Conference on Web Information Systems Engineering (WISE)*, Paphos, Cyprus, 2012.
10. Dorn C, Taylor RN. Analyzing runtime adaptability of collaboration patterns. In *International Conference on Collaboration Technologies and Systems (CTS), 2012, May 21-25, Denver, CO, IEEE*, Smari WW, Fox GC (eds). IEEE, 2012; 551–558.
11. Dorn C, Taylor RN. Mapping software architecture styles and collaboration patterns for engineering adaptive mixed systems. *Technical Report Technical Report*, Institute of Software Research, University of California: Irvine, 2011. Available from: http://www.isr.uci.edu/tech_reports/UCI-ISR-11-4.pdf [Accessed on 17 November 2014].
12. Dorn C, Taylor RN, Dustdar S. Flexible social workflows: collaborations as human architecture. *IEEE Internet Computing* 2012; **16**:72–77. DOI: 10.1109/MIC.2012.33.
13. Malone TW, Crowston K. The interdisciplinary study of coordination. *ACM Computing Surveys* 1994; **26**:87–119. DOI: 0.1145/174666.174668.
14. Georgas JC, Taylor RN. Policy-based architectural adaptation management: robotics domain case studies. In *Software Engineering for Self-Adaptive Systems*, Cheng BH, Lemos R, Giese H, Inverardi P, Magee J (eds). Springer-Verlag: Berlin, Heidelberg, 2009; 89–08. DOI: 10.1007/978-3-642-02161-9_5.
15. Grudin J. Why cscw applications fail: problems in the design and evaluation of organizational interfaces. *Proceedings of the 1988 ACM Conference on Computer-supported Cooperative Work*, CSCW '88, ACM, New York, NY, USA, 1988; 85–93. DOI: 10.1145/62266.62273.
16. ter Beek MH, Lenzini G, Petrocchi M. Team automata for security. *Electronic Notes in Theoretical Computer Science* 2005; **128**:105–119. DOI: 10.1016/j.entcs.2004.11.044.

17. Ellis CA, Gibbs SJ. Concurrency control in groupware systems. *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD '89, ACM, New York, NY, USA, 1989; 399–407. DOI: 10.1145/67544.66963.
18. Pacull F, Sandoz A, Schiper A. Duplex: a distributed collaborative editing environment in large scale. *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, ACM, New York, NY, USA, 1994; 165–173. DOI: 10.1145/192844.192900.
19. Kittur A, Kraut RE. Harnessing the wisdom of crowds in wikipedia: quality through coordination. *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, CSCW '08, ACM, New York, NY, USA, 2008; 37–46. DOI: 10.1145/1460563.1460572.
20. Brandes U, Kenis P, Lerner J, van Raaij D. Network analysis of collaboration structure in wikipedia. *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, ACM, New York, NY, USA, 2009; 731–740. DOI: 10.1145/1526709.1526808.
21. Eugster PT, Felber PA, Guerraoui R, Kermarrec AM. The many faces of publish/subscribe. *ACM Computing Surveys* 2003; **35**:114–131. DOI: 10.1145/857076.857078.
22. Grier D. A social history of bitnet and listserv, 1985–1991. *Annals of the History of Computing, IEEE* 2000; **22**(2): 32–41. DOI: 10.1109/85.841135.
23. Zhao D, Rosson MB. How and why people twitter: the role that micro-blogging plays in informal communication at work. *Proceedings of the ACM 2009 International Conference on Supporting Group Work*, GROUP '09, ACM, New York, NY, USA, 2009; 243–252. DOI: 10.1145/1531674.1531710.
24. Zhang J, Qu Y, Cody J, Wu Y. A case study of micro-blogging in the enterprise: use, value, and related issues. *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, CHI '10, ACM, New York, NY, USA, 2010; 123–132. DOI: 10.1145/1753326.1753346.
25. Adamic LA, Zhang J, Bakshy E, Ackerman MS. Knowledge sharing and yahoo answers: everyone knows something. *Proceeding of the 17th International Conference on World Wide Web*, WWW '08, ACM, New York, NY, USA, 2008; 665–674. DOI: 10.1145/1367497.1367587.
26. Lampe CA, Johnston E, Resnick P. Follow the reader: filtering comments on slashdot. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, ACM, New York, NY, USA, 2007; 1253–1262. DOI: 10.1145/1240624.1240815.
27. Nardi BA, Schiano DJ, Gumbrecht M, Swartz L. Why we blog. *Communications of the ACM* 2004; **47**:41–46. DOI: 10.1145/1035134.1035163.
28. Brabham DC. Crowdsourcing as a model for problem solving. *Convergence: The International Journal of Research into New Media Technologies* 2008; **14**(1):75–90. DOI: 10.1177/1354856507084420.
29. Callison-Burch C. Fast, cheap, and creative: evaluating translation quality using amazon's mechanical turk. *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, Association for Computational Linguistics, Stroudsburg, PA, USA, 2009; 286–295.
30. McAuley JJ, da Fontoura Costa L, Caetano TS. Rich-club phenomenon across complex network hierarchies. *Applied Physics Letters* 2007; **91**(8):084103. DOI: 10.1063/1.2773951.
31. Albert R, Jeong H, Barabási AL. The diameter of the World Wide Web. *CoRR* 1999; **cond-mat/9907038**:130–131.
32. Crowston K, Li Q, Wei K, Eseryel UY, Howison J. Self-organization of teams for free/libre open source software development. *Information and Software Technology* 2007; **49**:564–575. DOI: 10.1016/j.infsof.2007.02.004.
33. Ellis C, Nutt GJ. Workflow: the process spectrum. *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, State Botanical Garden of Georgia, 1996; 140–145.
34. van der Aalst WMP, ter Hofstede AHM, Weske M. Business process management: a survey. *Proceedings International Conference, BPM 2003 Eindhoven, The Netherlands, June 26-27, 2003, Lecture Notes in Computer Science 2678*, Springer Berlin Heidelberg, 2003; 1–12.
35. Cass AG, Lerner BS, Jr SMS, McCall EK, Wise AE, Osterweil LJ. Little-jil/juliette: a process definition language and interpreter. In *Proceedings of the 2000 International Conference on Software Engineering, 2000, June 4-11*, Ghezzi C, Jazayeri M, Wolf AL (eds). ACM: Limerick, Irland, 2000; 754–757.
36. Ford M, Endpoints A, Keller C. Ws-bpel extension for people (bpel4people), 2007. version 1.0.
37. Dustdar S, Hoffmann T. Interaction pattern detection in process oriented information systems. *Data & Knowledge Engineering* 2007; **62**:138–155. DOI: 10.1016/j.datak.2006.07.010.
38. Ouchi WG, Maguire MA. Organizational control: two functions. *Administrative Science Quarterly* 1975; **20**(4): 559–569.
39. Ben-Arieh D, Pollatscheck MA. Analysis of information flow in hierarchical organizations. *International Journal of Production Research* 2002; **40**(15):3561–3573. DOI: 10.1080/00207540210137611.
40. Künzle V, Weber B, Reichert M. Object-aware business processes: fundamental requirements and their support in existing approaches. *IJISMD* 2011; **2**(2):19–46.
41. Reijers H, Rigter J, Aalst WVD. The case handling case. *International Journal of Cooperative Information Systems* 2003; **12**:365–391.
42. Wilde E, Gaedke M. Web engineering revisited. In *VoCS'08 Proceedings of the 2008 International Conference on Visions of Computer Science: BCS International Academic Conference*, Erol Gelenbe, Samson Abramsky, Vladimiro Sassone (eds). British Computer Society Swinton: UK, 2008; 41–50.
43. Moreno N, Romero JR, Vallecillo A. An overview of model-driven web engineering and the mda. In *Web Engineering: Modelling and Implementing Web Applications*, Rossi G, Pastor O, Schwabe D, Olsina L (eds). Human-Computer Interaction Series, Springer: London, 2008; 353–382. DOI: 10.1007/978-1-84628-923-1_12.

44. Schwinger W, Retschitzegger W, Schauerhuber A, Kappel G, Wimmer M, Pröll B, Castro Cachero C, Casteleyn S, De Troyer O, Fraternali P, Garrigos I, Garzotto F, Ginige A, Houben G, Koch N, Moreno N, Pastor O, Paolini P, Ferragud Pelechano V, Rossi G, Schwabe D, Tisi M, Vallecillo A, van der Sluijs K, Zhang G. A survey on web modeling approaches for ubiquitous web applications. *International Journal of Web Information Systems* 2008; **4**(3):234–305.
45. Ceri S, Daniel F, Matera M, Facca FM. Model-driven development of context-aware web applications. *ACM Transactions Internet Technology* 2007; **7**(1). DOI: 10.1145/1189740.1189742. Article No. 2.
46. Ceri S, Daniel F, Facca FM, Matera M. Model-driven engineering of active context-awareness. *World Wide Web* 2007; **10**:387–413. DOI: 10.1007/s11280-006-0014-5.
47. Gregg DG. Designing for collective intelligence. *Communications of the ACM* 2010; **53**:134–138. DOI: 10.1145/1721654.1721691.
48. Teruel MA, Navarro E, López-Jaquero V, Montero F, González P. Csrml: a goal-oriented approach to model requirements for collaborative systems. *Proceedings 30th International Conference, ER 2011, Brussels, Belgium, October 31 - November 3, 2011. Lecture Notes in Computer Science Volume 6998, 2011*, Springer, Berlin Heidelberg, 2011; 33–46.
49. Souza VES, Lapouchnian A, Mylopoulos J. System identification for adaptive software systems : a requirement engineering perspective. *System* 2011; **6998**:346–361.
50. Tamburri DA, di Nitto E, Lago P, van Vliet H. On the nature of the GSE organizational social structure: an empirical study. *Proceedings of the 7th IEEE International Conference on Global Software Engineering*, Porto Alegre, Rio Grande do Sul, Brazil, 2012; 114–123.
51. Tamburri DA, Lago P, van Vliet H. Organizational social structures for software engineering. *ACM Computing Surveys* 2013; **46**(1):1–34. Article No. 3.
52. Tamburri DA, Lago P, van Vliet H. Uncovering latent social communities in software development. *IEEE Software* 2013; **30**(1):29–36. DOI: 10.1109/MS.2012.170.
53. Doan A, Ramakrishnan R, Halevy AY. Crowdsourcing systems on the World-Wide Web. *Communications of the ACM* 2011; **54**:86–96. DOI: 10.1145/1924421.1924442.
54. Malone TW, Laubacher R, Dellarocas C. Harnessing crowds : mapping the genome of collective intelligence. *Technology* 2010; **1**(2):327–335.
55. Skopik F, Schall D, Dustdar S. Trusted interaction patterns in large-scale enterprise service networks. *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP '10, IEEE Computer Society, Washington, DC, USA, 2010; 367–374. DOI: 10.1109/PDP.2010.9.
56. Kittur A, Smus B, Kraut R. CrowdForge: crowdsourcing complex work. *Proceedings of the 2011 Annual Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, ACM, New York, NY, USA, 2011; 1801–1806. DOI: 10.1145/1979742.1979902.
57. Dustdar S, Gaedke M. The social routing principle. *Internet Computing, IEEE* 2011; **15**(4):80–83. DOI: 10.1109/MIC.2011.97.
58. Schall D, Skopik F, Psaier H, Dustdar S. Bridging socially-enhanced virtual communities. *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, ACM, New York, NY, USA, 2011; 792–799. DOI: 10.1145/1982185.1982356.
59. Kwak H, Lee C, Park H, Moon S. What is Twitter, a social network or a news media? *WWW '10: Proceedings of the 19th International Conference on World Wide Web*, ACM, New York, NY, USA, 2010; 591–600. DOI: 10.1145/1772690.1772751.
60. Ellis C. Team automata for groupware systems. *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: the Integration Challenge*, GROUP '97, ACM, New York, NY, USA, 1997; 415–424. DOI: 10.1145/266838.267363.
61. Kleijn J, Team automata for cscw - a survey. In *Petri Net Technology for communication-based systems - advances in Petri Nets, Lecture Notes in Computer Science*, Vol. 2472, Ehrig H, Reisig W, Rozenberg G, Weber H (eds). Springer, 2003; 295–320.