Abstract—Human collaboration has become an integral part of large-scale systems for massive online knowledge sharing, content distribution, and social networking. Maintenance of these complex systems, however, still relies on adaptation mechanisms that remain unaware of the prevailing user collaboration patterns. Consequently, a system cannot react to changes in the interaction behavior thereby impeding the collaboration's evolution. In this paper, we make the case for a human architecture model and its mapping onto software architecture elements as fundamental building blocks for system adaptation.

Keywords—collaboration patterns, software architecture, adaptation, context awareness

I. INTRODUCTION

Some of the most successful endeavors in the recent years built upon large-scale Web systems coupled with massive user participation. Unprecedented forms of knowledge creation (e.g., Wikipedia), source code production (e.g., GitHub, SourceForge), knowledge sharing (e.g., Yahoo! Answers), news distribution (e.g., Twitter), or social networking (e.g., Facebook, LinkedIn) owe their existence to user-driven collaboration, and user-generated content, respectively. Humans increasingly become provider and consumer of content and computation. As the report on Ultra-Large-Scale (ULS) Systems [1] highlights: people are no longer just the “users” of a system but an integral part. Such socio-technical systems have become too complex and too large-scale to be managed by human administrators alone and mechanisms for self-adaptive systems have been proposed [2]. In this respect, the ULS report further emphasizes that social interactions are highly relevant to the design and adaptation of ULS systems.

Current adaptation mechanisms, however, consider only the software system and remain unaware of the implications arising from collaboration interdependencies. A system neglecting the collective behavior is unable to support efficient operation and evolution of collaborative efforts. The above mentioned systems lack collaboration-aware adaptation capabilities and thus cannot provide but the simplest interaction forms. Harnessing the wisdom of the crowd for problems that require complex coordination remains out of reach.

As repeatedly pointed out [3], [4], architecture-based approaches to self-management address adaptation on the "right" level of abstraction and generality, rather than focusing on language-level or network-level adaptation. On the software architectural level, adaptation actions typically describe the replacement of components and the reconfiguration of connectors. Existing research in the domains of architecture description languages (ADL) [5], [6], architecture-based design [7], [8], or architecture-driven adaptation [9] remains unconcerned by implications of the collaboration structure. We propose to apply the notion of architecture-based adaptation to the collaboration level. Our human architecture model describes collaborations in terms of human components and human connectors. The realization of these collaboration patterns through software elements is subsequently detailed through linking of the software architecture model and the human architecture model. The deployed software system references both models and thus allows an adaptation mechanism to reason upon the effect of software changes on the collaboration structure. Similarly, human architecture changes drive software architecture adaptations to prepare for changing user behavior.

II. MOTIVATING EXAMPLE

Monitoring and safety systems range in scope from a small security team handling an office building to thousands of personnel in back offices and on site at geographically distributed locations to secure critical infrastructure. These systems tightly interweave people and software components and hence need co-adaptation of collaboration structures and software architectures. In the building monitoring case, back office operators utilize high definition video streams, floor plans, building sensor feeds, occupancy logs, and communication channels with on-site security staff. Reassigning tasks amongst team members, reacting to non-responding team members, or adding new team members are examples of collaboration-driven adaptations that result in reconfiguration of the underlying software structure. The adaptation mechanism deploys new video relay components in proximity to the new subscriber(s), installs additional client components, and updates event subscriptions. Simultaneously, the monitoring system needs to react to software-level events such as failing components, congested data links, and emergence of new information sources. Collaboration-aware software architecture adaptations include mitigating
bandwidth limitations by reducing a video-stream’s quality (or even dropping it) amongst co-located operators while assuring that at least one shared high-definition stream remains available. Adaptations also involve delegating changes to the collaboration level such as assigning more humans when the amount of incoming events threatens to overwhelm the current team.

III. APPROACH

In software architectures, components are the loci of computation and data management whereas connectors facilitate and control the interactions between components. We can likewise distinguish between humans according to work-focused and coordination-focused roles. We thus describe a human architecture in terms of human components and human connectors (more details in Section IV).

Figure 1 summarizes the interdependencies between human and software architecture, software components, and the adaptation mechanism. The Design-time Human Architecture specifies collaboration patterns such as Master/Worker, Secretary/Principal, or Organizational Control [10]. The corresponding Design-time Software Architecture details the individual software elements that realize the collaboration patterns. A mapping from human to software architecture enables the annotation of executable Software Components with collaboration patterns and software component types. Upon deployment, the Runtime Human Architecture describes the actually involved people and their roles whereas the Runtime Software Architecture determines the Deployed SW System. The Adaptation Manager receives software level events, utilizes the runtime architecture models for reasoning on collaboration-aware reconfigurations, and, in return, keeps the models in sync with the deployed system.

IV. LINKING HUMAN AND SOFTWARE ARCHITECTURE

The core human architecture elements are HumanComponents, HumanConnectors, and CollaborationObjects (Fig. 2 top left). CollaborationObjects in the form of messages, streams, and shared artifacts serve as the means of human interaction. HumanAction and ObjectAction define access privileges (create, read, write, delete) on CollaborationObjects. Connections link multiple individual components, connectors, and objects through their actions to establish collaboration patterns. At runtime, RolePrivileges assign HumanComponent and HumanConnector capabilities to ImplActors (i.e., user, software agent, skill, or group). A reference to a CollaborationObject scopes the privileges to interactions amongst a set of ImplActors. Ultimately, the CollaborationObjectInstance connects multiple ImplActors in their particular RolePrivileges, thereby describing actual interactions. We implemented the human architecture model in the Generic Modeling Environment1 (GME). On the software architecture level, we chose the existing eXtensible Architecture Description Language (xADL) [6]. In xADL, ComponentTypes and ConnectorTypes exhibit Signatures that signal compatibility when pointing to the same InterfaceType (Fig. 2 lower left). At runtime, Links define the instantiated Component and Connector topology.

We need to capture correspondence between software and collaboration elements. The Cross-Model Links detail which ComponentType (and substructure(s), not shown in Fig 2 for sake of clarity) realizes a particular HumanComponent. Dedicated ComponentTypes manage access to and delivery of CollaborationObjects. The HumanActions and ObjectActions correspond to InterfaceTypes. A (human level) Connection hints at the ComponentTypes that need to interact and thus maps to a set of ConnectorTypes. At runtime we merely need to trace which Component serves a specific ImplActor, respectively manages a particular CollaborationObjectInstance. Note that mappings are not necessarily one-to-one (e.g., a ComponentType potentially manages multiple CollaborationObjects, while a single CollabObjectInstance might be replicated across multiple Components). A system adaptation mechanism subsequently exploits these Cross-Model Links to become collaboration aware.

V. COLLABORATION AND SOFTWARE CO-ADAPTATION

The proposed Adaptation Manager (Fig. 3) combines two feedback cycles to realize co-adaptation of the collaboration structure and the actual software system. The common adaptation knowledge includes human and software architecture, Quality of Service (QoS) requirements, and domain specific architecture constraints and resource limitations (e.g., # of staff members, # of available mobile cameras, connector bandwidth).

\footnote{\url{http://www.isis.vanderbilt.edu/Projects/gme/}}
Collaboration-centric raw events such as task timeouts trigger **Collaboration Monitoring** (1) to detect high-level symptoms (e.g., the user missed a final task deadline). **Collaboration Analysis** (2a) then determines suitable mitigation actions and issues a change request (e.g., reassign the task). **Collaboration Planning** (3) transforms the change request into a change plan that outlines how to reconfigure the **Runtime Human Architecture** accordingly (e.g., removing a link, assigning a role to a particular user). **Collaboration Execution** implements the collaboration changes in the **Runtime Human Architecture** (4a) and issues corresponding minimal **Runtime Software Architecture** reconfigurations (4b). The subsequently triggered (4c) **Software Analysis** establishes which additional software changes (5) need to be made to maintain the QoS requirements given the underlying resources and structural constraints. An example change request that considers collaboration structure demands replication of a video relay component to ensure video streaming reliability. **Software Planning** (6) creates the appropriate plan and has **Software Execution** carry out all final software changes (7a) via **Effectors**. Software Execution also ensures (7b) that the software architecture reflects these additional changes.

Software-centric raw events like failing message calls enable **Software Monitoring** (i) to detect a failing component (e.g., mobile camera out of battery). **Software Analysis** (ii) assesses the impact on the collaboration structure, requests countermeasures (as before, 5), and updates available resources and system constraints (iii).

**VI. Discussion**

Distinct human and software models are a necessity for co-adaptation. Human collaboration patterns are too independent from the underlying software topology to reduce them to annotations or simple aspects of the system architectures. The collaboration structure is directly available during runtime instead of having to extract it from the (typically more fine-grained) software architecture. Reconfigurations at the collaboration level rarely correspond to structurally equivalent changes at the software level and vice versa. Hence, collaboration changes would stay unnoticed in the software structure, while software topology modifications would require additional analysis whether the human architecture remained the same. Having two separate models, however, comes at a cost. Keeping the models synchronized requires extra effort and discipline. We expect runtime mechanisms such as software sensors to provide the required information on actual ongoing collaborations and their links to software elements. Especially large-scale deployments will have to increasingly rely on users self-managing their (own) role privileges (where applicable). This requires a rethinking when designing and managing large-scale socio-technical systems. The co-adaptation approach presented here constitutes a first attempt, but a set of open challenges remains. Incorporating hidden communications channels and reliably observing physical interactions demands further investigations. The current approach is domain agnostic. We suggest addressing for each application domain independently (a) how to adapt inherently unreliable human behavior (which ranges from non-intrusive information ranking or recommendations, to automatic actions as in Amazon MTurk) and (b) how to trade-off privacy concerns incurred by monitoring collaborations. In addition, large-scale systems challenge the feasibility of centralized adaptation. Existing research on adaptation scopes and hierarchical feedback loops provides insights in how to decentralize the Adaptation Manager.
VII. RELATED WORK AND CONCLUSIONS

Research in autonomic computing and self-adaptive systems aims to reduce the necessary user involvement in system management [11], [2]. Architecture-driven adaptation techniques are amongst the earliest [3] and continuously relevant approaches [12] as demonstrated by successful application to mobile environments [13], robotics systems [9], and adaptive service compositions [14]. Architecture description languages (ADL) [5], [6] and architecture-based design [7], [8] in particular, and autonomic computing research efforts in general, focus on pure technical issues. User context gained more attention only recently [2]. The implications of the collaboration structure, however, remain so far unaddressed.

Recent work on large-scale Web systems concentrates on the general Web 2.0 idiosyncrasies but neglects specific interaction structures at runtime [15]. Similar to research in autonomic computing, model-driven Web engineering approaches focus primarily on software aspects [16], [17] but don’t go beyond (user) context-centric adaptations [18].

To the best of our knowledge, this is the first attempt to combine human and software architectures for collaboration-aware self-adaptation. We provided a design-time and runtime model for capturing collaboration patterns and the subsequent mapping onto software architecture elements. The current models provide a barebone construct. We suggest adding domain-specific properties as needed for particular adaptation purposes through the extension mechanisms of GME and xADL, and as part of our future work. Our immediate next efforts will focus on coupling collaboration and software adaptation cycles through automatically deriving software architecture reconfigurations from collaboration changes.

Acknowledgment

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

References


