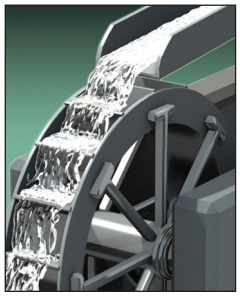
Editor: Schahram Dustdar • dustdar@infosys.tuwien.ac.at



Flexible Social Workflows

Collaborations as Human Architecture

Christoph Dorn and Richard N. Taylor • *University of California, Irvine*

Schahram Dustdar • *Technical University of Vienna*

Human process involvement has gained momentum in recent years, but the proposed mechanisms can't efficiently adapt Web-scale collaborative workflows. Here, the authors describe collaborative problem solving and its integration with process-support systems as an architecture comprising human components and connectors. This modeling of coordination and execution roles enables reasoning on workflow flexibility and appropriate adaptation actions.

In recent years, the business process community has realized the importance of collaboration in achieving flexible and adaptive workflow systems.[1] Typical workflows are rigidly designed to achieve reproducible, reliable process outcomes. Consequently, such workflows restrict human involvement to limit unpredictable behavior and side effects. Existing workflow languages such as BPEL4People and crowdsourcing environments such as Amazon Mechanical Turk don't foresee collaboration among process participants and thus can't harness the power of collaborative problem solving.

Designing and deploying processes beyond organizational structures' traditional boundaries requires that we rethink how to achieve process reliability and control. Individual human involvement can never guarantee 100 percent success. The opportunities for failure are considerably higher when multiple humans must collaborate to produce a common outcome. How, then, can we design processes to gracefully handle failures and unexpected situations in massively collaborative business processes?

Here, we highlight two major obstacles: First, beyond simple process awareness, most process frameworks offer limited support for coordination patterns among multiple collaborating participants. Second, current process adaptation mechanisms remain unaware of underlying collaboration patterns.

Lacking the appropriate patterns either leads to process default (as when participants can't complete the task), or participants resort to generic communication forms outside the process engine's realm. External mechanisms can't provide process awareness support to participants and remain ignorant of events relevant to the process engine. Even with the right patterns in place, the process adaptation manager must still become aware of the actual collaboration structure. Failing to do so seriously impedes the manager's ability to reason about process progress and applicable adaptation actions.

We propose going beyond the simple mechanisms that prescribe current human process involvement and describe human coordination and execution roles in the form of a *human architecture*. We're inspired by software architecture languages (ADLs) that describe a software system's design in terms of *components* (the loci of computation and data management) and *connectors* (which facilitate and control the interactions between components). Following this perspective, a traditional process engine resembles a connector that manages task, data, and temporal dependencies between executing components (human and software).

Published by the IEEE Computer Society

1089-7801/12/$31.00 © 2012 IEEE

Collaborations as Human Architectures

When we compose collaboration patterns from (human) components and (human) connectors, we can precisely define how much flexibility we want to grant humans in shaping a process during runtime. Rigid workflows keep all coordination privileges with the process engine and limit humans and services to executing computational tasks. On the other extreme end, ad hoc workflow systems offer full coordination capabilities to process participants but lack automated support. Among the myriad collaborative process-support systems, none makes the fundamental distinction between (human) components and (human) connectors.

This distinction, however, is imperative to determine how flexible and adaptive a collaboration structure is. In software architectures, connectors are the key element to system adaptability. For example, connectors allow an adaptation mechanism to dynamically replace behavior components in robotic systems without affecting other components.[2] Web proxies are connectors on the Internet that decide which server (component) should process a particular client (component) request. Overloaded or unavailable servers thus become transparent to the client. In human collaborations, a secretary (connector) might respond on behalf of the principal (component) or coordinate meeting room reservations, thereby managing access to a shared resource (component).[3] The importance of collaboration connectors grows with joint efforts' scale and complexity, especially in distributed settings where individual collaborators have little opportunity for informal communication.

Example Scenario

Consider the following collaborative process for producing a final report in a large research project. At minimum, Alice, the scientific project leader, must carry out the following steps:

1. get hold of all relevant authors,
2. collect their contributions, and
3. format and quality check the aggregated report (see Figure 1).

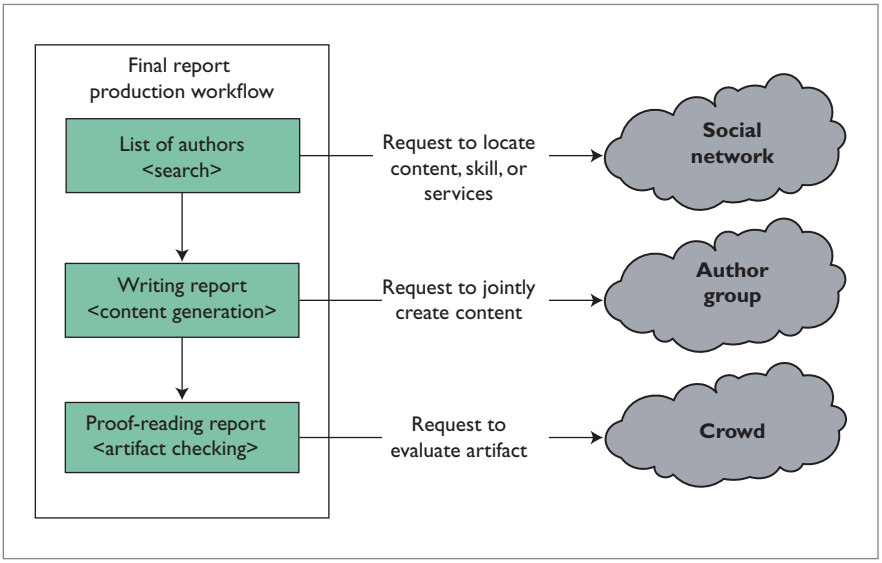


Figure 1. Scenario workflow. Peer-to-peer-style search requests target the social network, a shared artifact coordinates collaborative content generation, and the crowd handles well-defined, independent tasks.

Each step calls for a different collaboration pattern and thus comes with particular adaptation idiosyncrasies.[4]

In a large research project, Alice isn't likely to know all relevant authors directly. Instead, she initiates a peer-to-peer (P2P)-style request through her social project network to identify potential experts. Then, Alice collects the experts' contributions. Instead of sending individual content items, Alice coordinates authors through a set of wiki pages. The wiki serves as a shared artifact to track progress, achieve awareness among authors, and relieve Alice of detecting and handling conflicts.

The compiled report requires spell checking and final formatting before submission. A final report easily comprises hundreds of pages, which would overwhelm a single person. For this task, Alice decides to break the report into multiple parts and have multiple workers check each part separately. Such uniform, independent jobs are best suited for crowdsourcing.

As mentioned, each task requires a particular collaboration pattern that differs in terms of participant coupling, communication form, outcome, and adaptation constraints. When designing mechanisms for process adaptation, we must be aware of these pattern-specific implications.

The BASE Framework for Collaboration Patterns

Besides basic concepts, the software architecture domain also provides a framework for analyzing collaboration patterns' flexibility. We apply the software runtime adaptation aspects defined in the BASE framework — *behavior*, *asynchrony*, *state*, and *execution context*.[5]

Behavior focuses on the means of adaptation and the scope of supported change. Collaboration behavior adaptation includes rewiring team members (adding or removing coworker relations in a social network), replacing them (exchanging a report coauthor), or reassigning a task (because the crowd fails to

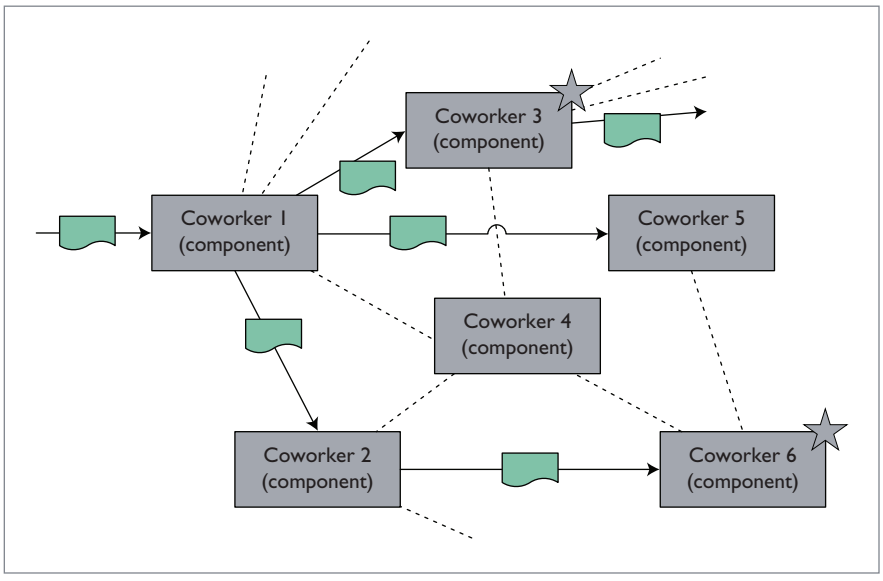


Figure 2. Social network collaboration pattern. Here, we apply a social network for peer-to-peer-style information retrieval. Arrows represent forwarded requests, dotted lines are unused coworker links, and stars mark relevant contacts.

submit a quality report, for example). Messages between members and shared artifacts represent significant loci of adaptation. A new message type might introduce an explicit reply-by date. A pattern further specifies whether adaptation is limited to the composition of existing behaviors or if the adaptation mechanism can introduce new ones, and what behavior must remain unmodified. For example, in the master/worker pattern (the generic form of crowdsourcing), replacing a worker is comparatively simple, but replacing the assignment connector is extremely difficult.

Asynchrony raises awareness of temporal adaptation implications. Large-scale collaborations take potentially longer to update than compact teams and might never reach a completely updated status. The lag between initiating an adaptation plan and its completion raises questions concerning constraints that the adaptation mechanism must enforce during adaptation. For example, when replacing a document's authors, one previous coauthor must remain available during the transition phase (rather than exchanging all authors at once).

State refers to potential adaptation side effects when we alter the communication method, manipulate shared artifacts, or replace workers. The most knowledgeable form of direct state change is loss of implicit collaboration know-how upon removing a worker. When adapting the human interaction structure, we must explicitly consider the handover of such implicit collaboration information between outgoing and incoming workers. Coauthoring a report requires less state management when coordinating through a shared artifact than when sharing draft versions directly among authors.

Execution context raises awareness of constraints that determine whether and when to adapt. Multiple factors such as explicit contracts, cost and time for repeating a task, and the execution of compensation actions influence the decision to adapt during an active collaboration session (for instance, if a human can cease work on a particular task or must wait until task completion). Replacing a coauthor while working on a shared document comes with higher transition costs than reassigning a self-contained job to another worker in the crowd.

Applying BASE to Example Patterns

In our scenario, we identify three collaboration patterns: the social network acts as a P2P network, the wiki provides asynchronous, decentralized coordination through shared artifacts, and crowdsourcing mimics parallel computing.

Social Networks

Software P2P systems primarily serve to locate and retrieve content. Several functionally identical peers — each maintaining a limited list of neighboring peers — forward requests that they can't locally fulfill. The major motivation for using P2P is resilience to node failures. To this end, P2P systems usually lack a centralized authority and expect peers to arbitrarily join and leave the network.

In our scenario, Alice applies the principle of P2P to social networks by issuing a request for project members to contribute to the final report. Her contacts within the project scope subsequently forward her request within their networks. The social network's structure ensures that Alice reaches all relevant participants even when some contacts fail to forward or misaddress the request (see Figure 2).

Behavior. Social networks are typically self-organizing; members join freely and form arbitrary links with new acquaintances. In work environments, social networks let individual employees form ties with relevant coworkers rather than rely merely on top-down designed and inflexible organization charts. Task- or project-specific social networks offer more control to managers, who decide on individual employee participation.

Nevertheless, employees still control their friend structures.

Asynchrony. Typically, only a (small) subset of members fluctuates at any single point in time — members go offline during vacations, temporarily work on different projects, or leave the organization for other jobs. Fluctuations, however, hardly disrupt the underlying social structure in the presence of sufficient ties. Thus, P2P-style requests needn't wait for a stable social network.

State. Collaboration know-how is generally spread across multiple members. In the scenario we describe, all researchers involved in a joint activity replicate that collaboration state subset. Thus, new members need support in connecting with relevant existing team members to gather state information before they can respond to P2P style requests themselves.

Execution context. Adaptation actions such as fluctuating members and the rewiring of "friend links" are independent from request processing and forwarding when such requests require only minimal time or effort. In our scenario, a coworker decides to reply to, forward, or ignore the request for authors. More time-consuming requests such as creating report content require the coworker to remain active or delay completion.

Shared Artifact

The P2P style is suitable for finding the right report contributors, but to coordinate the actual authoring, Alice prefers to apply shared artifacts. Coauthors require task awareness to avoid duplicating work efforts while asynchronously working on their contributions.

Shared artifacts such as wiki pages decouple producers from consumers. Collaborators obtain write access to manipulate the shared artifacts. Updates become visible to any interested participant (see Figure 3).

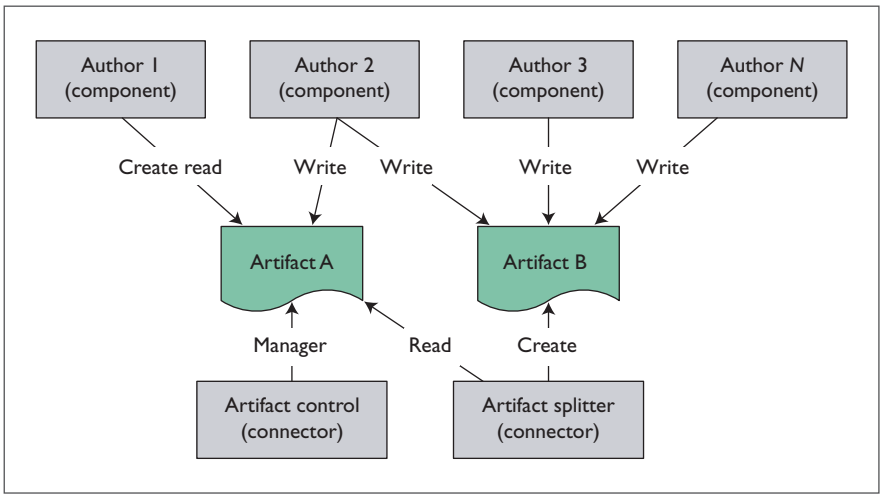


Figure 3. The shared artifact collaboration pattern. Here, multiple authors coordinate via shared artifacts and (human or software) connectors.

Behavior. Collaborators may join or leave the workspace at any time. They're free to create new shared artifacts or manipulate existing ones without requiring a dedicated person who collects, merges, and distributes contributions. Artifacts might be split or merged for performance reasons. Alice splits the project report as individual sections become too large and too heavily edited. Authors inspect the various parts and then decide what to work on without having to coordinate with all other coworkers directly.

Asynchrony. Adaptation actions occur on an artifact basis, affecting only a subset of all contributors. When Alice decides to split a particular section, only the involved authors perceive this change.

State. The shared artifact maintains the collaboration status. Collaborators construct their internal states from the artifact's history. Late-joining coauthors immediately notice who's contributed which sections and what parts are still missing.

Execution context. Collaborators must complete their artifact update before it's split or merged. Small, self-contained report changes allow for timely artifact adaptations. Locking mechanisms should be in place to allow a single author to update large parts of the artifact without creating conflicts.

Crowdsourcing

Proof-reading the final report requires almost no coordination among participants, involves a clear set of skills, and is highly repetitive. Thus, Alice (the master) leverages parallel execution, dividing the report into multiple independent proof-reading tasks, as in Map-Reduce. Task distribution uses both push and pull styles. In the former case, a *job advertising connector* replicates the task and assigns these replications as jobs directly to workers, whereas in the latter case, workers choose which jobs they prefer to work on (see Figure 4). Crowdsourcing is a specific form of the master/worker pattern in which job assignment is only pull-based.

Behavior. Here, the master decides how many workers can work in parallel on the same task artifact. In the pull-style assignment, workers choose which job to perform and whether to return a job unfinished.

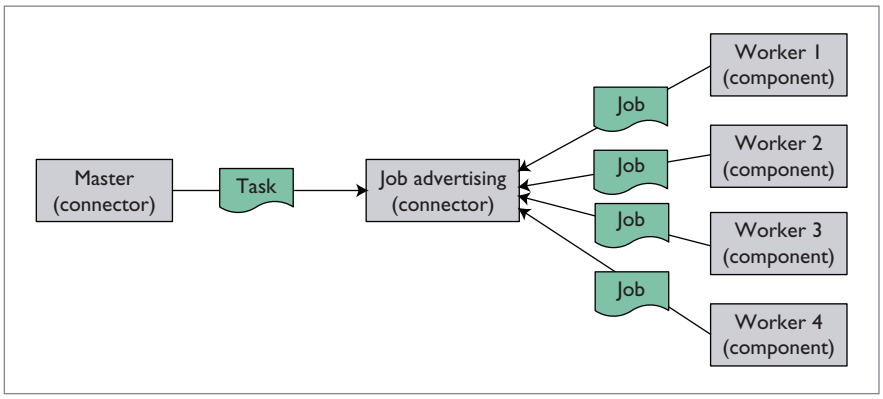


Figure 4. The crowdsourcing collaboration pattern. Here, a single task splits into multiple independent, replicated jobs that workers then claim.

In the push-style assignment, workers receive new jobs in their inbox but might still have the option to reject or delegate a job. Alice posts the quality-checking tasks on Amazon Mechanical Turk and restricts them to workers with an appropriate "English language" skill.

Asynchrony. A task artifact completely decouples the master from the workers. The job advertising connector reallocates a job to another worker when the initial worker fails to complete the task in a predefined timeframe. The master posts the task again, when the job results are of insufficient quality.

State. The task artifact contains the complete collaboration state. Replacing workers has no side-effect on the state. Multiple workers thus check each report section in parallel but have no impact on each other's job execution because no interaction is possible among workers.

Execution context. Multiple workers assigned to the same task artifact work on distinct copies and have no knowledge about each other. They remain similarly unaware if the master is replaced. A new worker simply obtains the task description and commences task execution independently of any previous work done.

Pattern-Specific Process Support

No single collaboration pattern would support all three process steps equally well. Likewise, a process-support system needs specific monitoring and adaptation strategies in each case. Figure 5 provides a screenshot from our human architecture modeling environment (based on the Generic Modeling Environment; www.isis.vanderbilt.edu/Projects/gme). Collaboration patterns comprise *HumanComponents*, *HumanConnectors*, and *CollaborationObjects*. Each element defines a set of collaboration actions (for example, create, observe, or claim) through which components and connectors are linked to collaboration objects (such as messages, artifacts, or streams). Adding elements such as *observer* components improves the adaptability of generic collaboration patterns such as P2P, shared artifact, and master/worker.

In the P2P pattern, observing forwarded messages lets the process engine perceive how far the request has spread and when to stop waiting for outstanding replies. Example short-term adaptation strategies include recommending new contacts, provisioning specific message types to improve collaboration efficiency, and repeating requests to counteract ignored messages.

For the shared artifact pattern, the process engine observes the number of involved authors, whether multiple authors engage in an edit war, and the frequency of write requests. Adaptation actions address the applicable mechanisms for conflict detection, conflict avoidance, and content-merging capabilities. Large, heavily edited artifacts become candidates for splitting.

Finally, the crowdsourced spell-checking task requires the process engine to monitor the successful completion of individual jobs. The process engine schedules multiple identical tasks for the sake of reliability, or issues multiple sequential tasks until it receives the desired results.

These examples are for pattern-specific observation and adaptation behavior. Adaptation strategies potentially leverage cross-pattern synergies. If some author doesn't respond, and we know that we applied the P2P pattern to find him, we subsequently explore the social network to find a suitable replacement. The same strategy doesn't work when the author is a worker from the crowd.

The explicit distinction between (human) components and (human) connectors is just the first step in achieving adaptive, Web-scale collaborative workflows. Much effort needs to go into investigating how the applied patterns should change at runtime, how to integrate humans and software within a collaboration-level connector, and how to achieve a balance between autonomic and recommendation-driven adaptations.

References

1. S. Dustdar and M. Gaedke, "The Social Routing Principle," *IEEE Internet Computing*, vol. 15, no. 4, 2011, pp. 80–83.
2. J.C. Georgas and R.N. Taylor, "Policy-Based Architectural Adaptation Management: Robotics Domain Case Studies," *Software Eng. for Self-Adaptive Systems*, B.H. Cheng et al., eds., Springer, 2009, pp. 89–108.

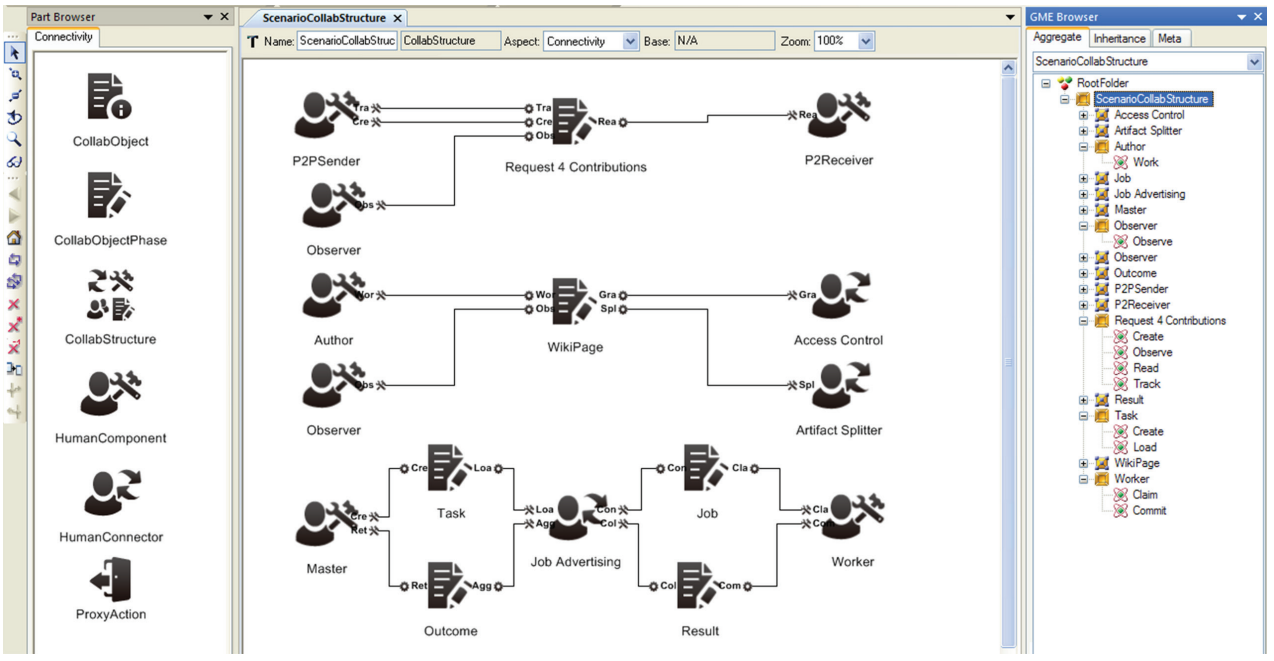


Figure 5. The human architecture modeling environment. A software designer models the desired collaboration patterns based on HumanComponents, HumanConnectors, and CollaborationObjects.

3. S. Dustdar and T. Hoffmann, "Interaction Pattern Detection in Process-Oriented Information Systems," *Data Knowledge Eng.*, vol. 62, July 2007, pp. 138–155.
4. C. Dorn and R.N. Taylor, *Mapping Software Architecture Styles and Collaboration Patterns for Engineering Adaptive Mixed Systems*, tech. report, Inst. of Software Research, Univ. of California, Irvine, June 2011; www.isr.uci.edu/tech_reports/UCI-ISR-11-4.pdf.
5. R.N. Taylor, N. Medvidovic, and P. Oreizy, "Architectural Styles for Runtime Software Adaptation," *Proc. Joint Working IEEE/IFIP & European Conf. Software Architecture* (WICSA/ECSA), IEEE Press, 2009, pp. 171–180.

Christoph Dorn is a visiting researcher at the Institute for Software Research, University of California, Irvine. His current research focuses on co-adaptation of software systems and human collaboration. Dorn has a PhD in computer science from the Vienna University of Technology. He was recently awarded an Austrian Science Fond (FWF) Schroedinger Mobility Fellowship. Contact him at cdorn@uci.edu; http://christophdorn.wordpress.com.

Richard N. Taylor is a full professor of information and computer sciences at the University of California, Irvine, and the director of the Institute for Software Research, which is dedicated to fostering innovative basic and applied research in software and information technologies. Taylor has a PhD in computer science from the University Colorado at Boulder. He's an ACM fellow and received the 2009 ACM SIGSOFT Outstanding Research Award. Contact him at taylor@uci.edu; www.isr.uci.edu/~taylor/.

Schahram Dustdar is a full professor of computer science (informatics) with a focus on Internet technologies and heads the Distributed Systems Group, Institute of Information Systems, at the Vienna University of Technology (TU Wien). Dustdar is an ACM Distinguished Scientist. Contact him at dustdar@infosys.tuwien.ac.at; www.infosys.tuwien.ac.at/.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.