Analyzing Design Tradeoffs in Large-Scale Socio-technical Systems through Simulation of Dynamic Collaboration Patterns

Christoph Dorn[1], George Edwards[2], and Nenad Medvidovic[3]

[1] Institute for Software Research, University of California, Irvine, CA 92697-3455
cdorn@uci.edu
[2] Blue Cell Software, Los Angeles, CA 90069, USA
george@bluecellsoftware.com
[3] Computer Science Department, University of Southern California, Los Angeles, CA 90089, USA
neno@usc.edu

Abstract. Emerging online collaboration platforms such as Wikipedia, Twitter, or Facebook provide the foundation for socio-technical systems where humans have become both content consumer and provider. Existing software engineering tools and techniques support the system engineer in designing and assessing the technical infrastructure. Little research, however, addresses the engineer's need for understanding the overall socio-technical system behavior. The effect of fundamental design decisions becomes quickly unpredictable as multiple collaboration patterns become integrated into a single system.

We propose the simulation of human and software elements at the collaboration level. We aim for detecting and evaluating undesirable system behavior such as users experiencing repeated update conflicts or software components becoming overloaded. To this end, this paper contributes (i) a language and (ii) methodology for specifying and simulating large-scale collaboration structures, (iii) example individual and aggregated pattern simulations, and (iv) evaluation of the overall approach.

Keywords: Design Tools and Techniques, System Simulation, Collaboration Patterns, Large-scale Socio-Technical Systems.

1 Introduction

During the last two decades, numerous web-based, large-scale collaboration services have emerged for social networking (e.g., Facebook), collaborative tagging (e.g., Digg), content sharing (e.g., Youtube), knowledge creation (e.g., Wikipedia), crowdsourcing (e.g., Amazon Mechanical Turk), and source code production (e.g., GitHub). Recent research efforts have analyzed these systems in terms of user incentives, participation, recruitment, decision making, and information management [18,5]. Engineered for diverse purposes, these systems differ widely in the underlying collaboration patterns of their users [6]. For example, Amazon

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 362–379, 2012.

Mechanical Turk follows the *master/worker* pattern for task outsourcing, Facebook links people in *peer-to-peer* social networks, Wikipedia manages *shared artifacts* for collaborative editing, and Twitter provides *publish/subscribe* news distribution.

Engineers of such systems have currently little support for anticipating the system's overall (i.e., socio-technical) behavior in large-scale deployments, in terms of metrics such as the timeliness and load of messages, artifact changes, queries, and so on. For example, a system architect for a knowledge creation platform might be interested in the number of write conflicts given particular contributor characteristics. Similarly, a crowdsourcing platform architect needs to consider effects of task assignment strategies on task response time. A microblogging system architect may want to estimate the event propagation speed for a given user subscription topology.

In general, an engineer aims to avoid negative behaviors such as information overload, decision making based on stale data, accidental information disclosure, or performance bottlenecks. These behaviors manifest themselves both within user collaborations and within the software itself. Subscribing to many Wiki articles, for example, may flood the user with updates and simultaneously overload the software that aggregates change events (information overload). On the other hand, a code repository user who is unaware of updates submitted by other users may encounter numerous burdensome write conflicts when submitting changes (information scarcity). The presence of multiple, aggregated patterns within a single system further complicates the problem, as complex interactions result in undesirable emergent behaviors that cannot be detected by observing individual patterns in isolation. System designers thus need sophisticated analysis to identify such undesirable behavior and the conditions that create it.

Once engineers understand the implications of particular combination of user behavior, collaboration patterns, and system configuration, they can apply system constraints at design-time that prevent the undesired effects completely or devise run-time adaptation mechanisms that mitigate those effects dynamically. The resulting systems are more robust and may feature more coordination and awareness mechanisms that are well-understood and governed. Without such support, systems may be brittle or restricted in terms of the provided collaboration mechanisms. For example, the successful mass-collaboration systems mentioned above apply limits for technical or collaborative reasons. [1] We are, however, interested in the effect of design decisions beyond simple constraints.

The primary technical challenge addressed in this paper is reasoning about the expected emergent behavior in large-scale collaborative systems prior to implementation and deployment. To this end, we propose *simulating the behavior of web-scale collaboration systems* in terms of collaborator structures, their actions, and the supporting software infrastructure. Several prior research efforts target important but only partial aspects of this problem, and thus fall short of delivering collaboration-centric design support. Existing work on simulating workflows or

[1] Facebook has an upper limit of 5000 friend connections, Twitter places an initial follow limit of 2000, and Wikipedia enforces rate limits on write requests.

crowdsourcing, for example, provides valuable insights into performance-improving algorithms [14,17,22], but addresses only a single subdomain. Simulations of software architectures and their implementation [8,2,15] focus on the software level rather than human interaction. Finally, in the domain of statistical mechanics, simulation and analysis of large-scale social networks remains very abstract [11,1]. General guidelines [12] for facilitating collaboration and user participation provide a starting point for designing large-scale systems. They, for example, recommend enabling users to edit and share data, but are insufficient for determining the specific effects within a given collaboration environment.

The primary contribution of this paper is a principled method for analyzing complex collaboration architectures through simulation. In support of this contribution, the paper describes:

- enhancements to an existing language for modeling collaboration patterns that enable dynamic analysis (Sec. 3),
- example models of several individual collaboration patterns as well as a model of their composition in a single system,
- specific techniques for scoping and targeting simulations to yield the most useful results, and
- an evaluation of the overall approach demonstrating its feasibility and usefulness (Sec. 4).

The following section provides a motivating scenario (Sec. 2); with related work in Section 5 and conclusion and outlook in Section 6.

2 Motivating Scenario

Building monitoring and security requires extensive collaboration among members of a security team. These teams range in scope from a small group that monitors an office building to hundreds of personnel in back offices and on-site that monitor critical, geographically distributed infrastructure. Facility monitoring systems that enable large-scale, flexible collaboration are subject to diverse coordination requirements. In this paper, we will illustrate key concepts based on such an example system composing collaboration patterns found in Twitter, Wikipedia, and Amazon Mechanical Turk.

Consider a large-scale facility monitoring system (Fig. 1) involving several different user roles: *sensors*, *field agents*, *back-office agents*, *back-office analysts*, and *team leaders*. Sensors may be hardware and software components or people with "eyes on the ground." Field agents are located on site and directly monitor data from sensors. Field agents are organized into teams assigned to a specific building, floor, or area. Field agents may send alerts of suspicious activity to back-office agents and flag relevant sensor data. Back-office agents investigate suspicious behavior by aggregating information and assessing whether a threat exists. Analysis of raw data from multiple video feeds, still images, and voice recordings may overwhelm an assigned back-office agent and require additional staff members on demand, in which case tasks can be assigned to a pool of back-office analysts. The team leader is responsible for determining the appropriate

response to an incident based on the aggregated, filtered information provided by back-office agents.

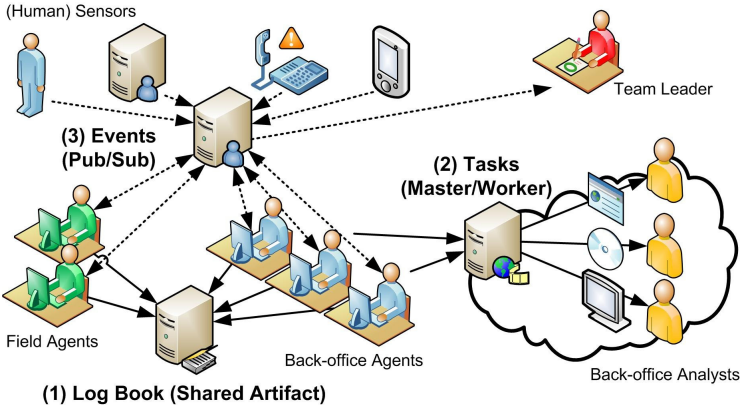

Fig. 1. Aggregating Collaboration Patterns for Infrastructure Monitoring

Interaction among the various users and user groups in this scenario exhibits several collaboration patterns. An engineer identifies following three potential collaboration patterns that match the underlying coordination requirements: **1**

Field agents will be provided with an interface for raising alerts and flagging sensor feeds that may indicate suspicious behavior. Back-office agents will select from a list of field agents from which they wish to receive alerts and flagged sensor data. Field agents need not be aware of which or how many back-office agents have selected to receive their alerts. Thus, information distribution will be achieved through the *publish/subscribe* collaboration pattern.

2 Virtual log books will be used to record the occurrence of events and user activities. Multiple log books may be created, each covering a different team, time-period, or subject. Users will be able to retrieve the latest log book on demand and make additions or modifications to it. Thus, the log books will implement the *shared artifact* collaboration pattern.

3 Back-office agents who require a threat assessment will add work items to a to-do list. Items in the to-do list will be automatically assigned to available back-office analysts, who will review the relevant sensor data and indicate whether a threat exists. In some cases, to minimize the potential for human error, the same task may be assigned to multiple analysts so that their conclusions can be compared for consistency. Thus, analysis tasks will be coordinated using the *master/worker* collaboration pattern.

The system design needs to achieve a balance of providing flexible, unrestricted collaboration mechanisms that facilitate staff members reacting to unforeseen situations while at the same time maintaining desirable system behavior. Staff members, for example, must not experience log book write conflicts, messages

and alerts need to be delivered to all interested parties but simultaneously avoid overloading the recipient, and tasks must finish in a timely manner and yield the required quality. The following questions highlight some specific issues the system designer might face:

1 How many field agents can a back-office agent reliably monitor before becoming overloaded? Should a limit be placed on how many field agents a back-office agent can select to monitor?

2 How should access to the log book be regulated to prevent write conflicts? Should staff members be required to obtain a lock before performing a write? If not, how often can we expect conflicts to occur?

3 How many users should share each log book? What happens if a large number of users are all trying to use the same log book?

4 How should tasks from the to-do list be allocated to available back-office analysts? First-in-first-out or some other way?

To answer all these questions, analysis of individual patterns is insufficient. The designer needs to consider system-wide, cascading implications such as the effect of event bursts on crowd-based situation assessment, the effect of overloaded crowd workers on timely event analysis, and the spike of write conflicts as staff members condense event observations into shared log books.

3 Modeling and Simulation of Collaboration Patterns

Large-scale collaboration systems heavily rely on humans as providers and consumers of information. Consequently, human behavior becomes an intrinsic aspect of the overall system. Given the inherent unpredictability of human behavior, static analysis techniques (e.g., [16]) are insufficient for making informed decisions about emergent behavior in a large-scale system. Instead, we focus on dynamic analysis in the form of system simulation.

In contrast to detailed modeling of software components, we propose simulating the interplay of humans and technology. In this context, the modeling effort focuses on collaboration patterns [6], such as master/worker, shared artifact, and publish/subscribe, rather than software architectures or architectural styles, such as SOA or 3-tier client-server [19]. Our approach treats collaboration patterns as "human architectures" consisting of people (*human components*) and the systems they use to facilitate collaboration (*collaboration connectors*). This achieves a clear distinction between work-centric roles (components) and coordination-centric roles (connectors), as described in our previous work [7]; emphasizes modeling of human interactions independently from the underlying design of the collaboration system used; and facilitates the identification of loci for system collaboration constraints.

Our process for simulating large-scale collaborations consists of three basic steps, usually performed in multiple iterations: (i) capturing collaboration patterns in an executable model, (ii) defining scenarios, assumptions, and configurations for individual simulation runs, and (iii) evaluating and interpreting

simulation results. These steps are described in Section 3.2. As a precondition, a suitable modeling language for capturing collaboration patterns must be specified. We have created such a language as an extension of the human Architecture Description Language (hADL [7], described in Section 3.1). This extended language is sufficiently flexible to allow engineers to model their own patterns or compositions of patterns, or they may extend the language further with pattern-specific elements or properties.

We used the DomainPro [2] modeling and simulation tool suite to create models of collaboration patterns that conform to the extended hADL language and run simulations of those models. DomainPro enables engineers to create custom simulation languages through metamodeling and supports agent-based and discrete event simulation semantics. In the following sections, the model diagrams and simulation shown were all developed in DomainPro. However, our overall approach is generic and could be easily applied using a different simulation environment.

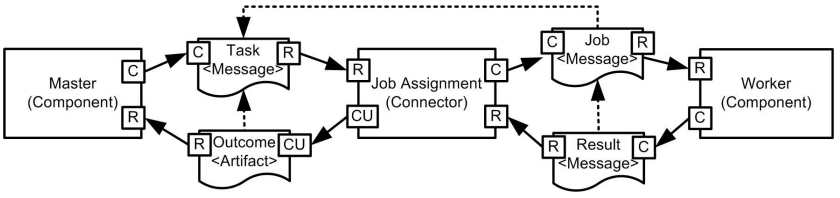

Fig. 2. Master/Worker pattern in hADL. Action labels represent CRUD privileges: **C**reate, **R**ead, **U**pdate, **D**elete. Collaboration objects exhibit optional references to related objects.

3.1 Modeling Language for Collaboration Patterns

In order to create simulation models for human collaboration, we extended an existing language, the human Architecture Description Language (hADL) [7], with additional features and properties to capture dynamic system behavior. As we will show, our extended hADL language can be used to capture a variety of individual and composite collaboration patterns.

We will briefly revisit the relevant parts of hADL as it represents the foundation for our simulation approach. In hADL, a collaboration pattern consists of two types of active entities: *human components* and *collaboration connectors*. A model of the master/worker pattern is shown in Fig. 2. The communication media used by components (e.g., Master, Worker) and connectors (e.g., Job Assignment) is represented by *collaboration objects*, such as messages (Task, Job, Result), streams, and artifacts (Outcome). *Human actions* and *object actions* restrict the interaction amongst components and connectors in terms of **C**reate, **R**ead, **U**pdate, and **D**elete manipulation capabilities. Connecting human actions and matching object actions gives rise to a collaboration pattern.

[2] `http://www.bluecellsoftware.com/`

Just as software-centric architecture description languages provide a high-level view of the system, hADL provides a view of human components, collaboration connectors, collaboration objects, and their wiring. However, execution of collaboration patterns remains outside of hADL's scope, and hADL does not address the interdependencies amongst multiple patterns in a complex collaboration system.

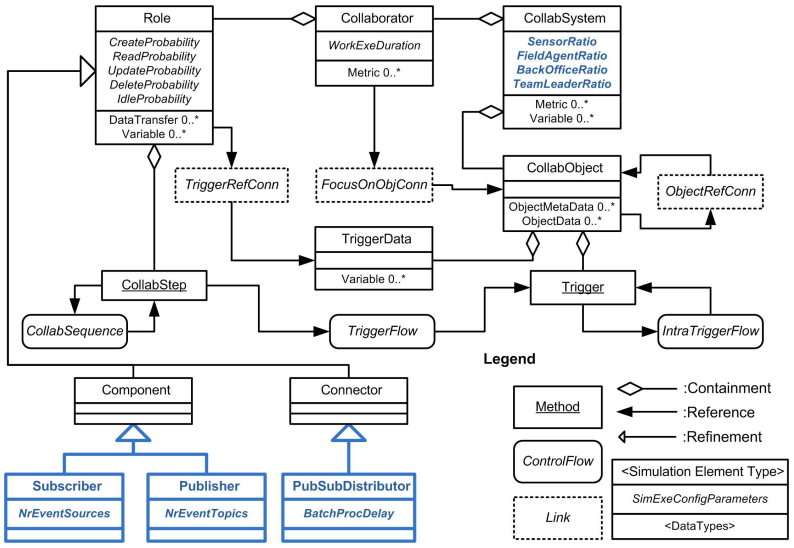

Fig. 3. The metamodel (extended hADL language) for defining collaboration patterns. Domain-specific extensions are highlighted in bold/blue.

Fig. 3 shows how hADL was extended to support dynamic behavior and simulation. A *CollabSystem* consists of *Collaborators*, which in turn play one or more *Roles*. A role may be either a Component or a Connector, thus enabling the Collaborator to assume a component role in one pattern and a connector role in another pattern. Each Role includes a set of *CollabSteps* that define the actions performed by the role. Each CollabStep represents a logical unit of work (e.g., sending a message, retrieving document content, processing a set of events). *CollabSequences* define the sequence (control flow) of CollabSteps. A CollabSystem also includes *CollabObjects*, the means of communication in hADL. *Triggers* represent events, such as such timer timeouts, that initiate collaborator responses. Additional links (i.e., *TriggerFlow*, *FocusOnObjConn*, *TriggerRefConn*, and *ObjectRefConn*) provide object references to complete the core model.

The types enumerated above contain properties (shown in italics) that can be varied to achieve different simulation behaviors. For example, CollabSteps have an associated duration (time needed to complete), and Roles define probabilities for engaging in different optional behaviors. These types and properties can

be further extended with pattern-specific types or properties if engineers wish to investigate additional aspects of a pattern. Fig. 3, for example, highlights additional Component and Connector subtypes as well as Configuration properties for the Publish/Subscribe pattern in bold/blue. Note that we excluded extensions for the other patterns used in the scenario and evaluation for sake of clarity.

3.2 Modeling Collaboration Patterns

To develop a model of a collaboration pattern or composition of patterns, an engineer may utilize our extended hADL language or a pattern-specific variant of it. The engineer defines the structure and behavior of the pattern(s) by instantiating the appropriate components and connectors and defining their interactions. The model usually also includes a set of data structures that encapsulate the system's dynamic state. Specifically, the simulation designer needs to identify:

Structure: Just as software engineering patterns [10] provide best-practises in programming, so do collaboration patterns [6] provide reusable structures of human components, collaboration connectors, actions, collaboration objects, and their relationships. Note that it is infeasible to explicitly model every Collaborator instance in large-scale systems. Hence, CollabObjects serve not only as carriers of information, but may also perform an addressing function (e.g., recording subscriptions to topics). Fig. 4 depicts the simulation model for a topic-centric publish/subscribe CollabSystem. The model combines both publisher and subscriber Roles within a single *Agent*. The *PubSubMW* Collaborator assumes the role of a collaboration connector for event delivery. *PubMsg* and *NfyMsg* provide the means of communication between Agent and PubSubMW.

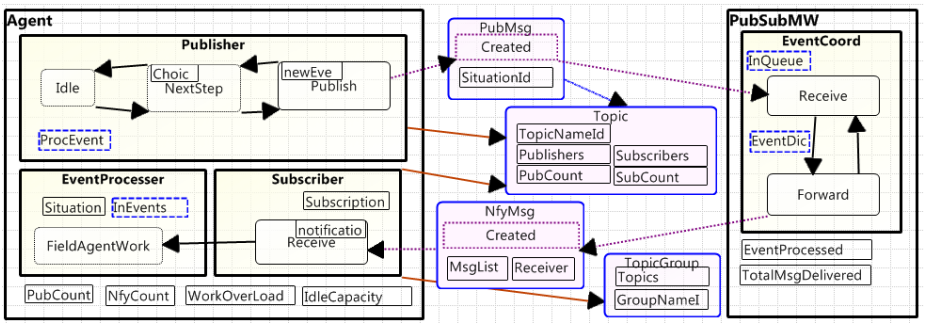

Fig. 4. Topic-centric Publish/Subscribe pattern in DomainPro Designer

Output: Behavior metrics provide engineers with information about the system's dynamic behavior. For example, the Agent workload when processing incoming events may be of interest to engineers. For the PubSubMW, engineers may be concerned with the number of notification messages/events processed during various batch processing intervals.

Logic: The overall system behavior emerges from the interactions among individual Collaborators' behavior as well as the control flow among them. To capture timing behavior, the simulation requires each CollabStep to specify its duration, which may be a stochastic or random value. In Fig. 4, CollabSteps such as Publish, Receive, and Forward contain the logic to create, read, and process collaboration object content. Data is transferred between CollabSteps either directly as input or indirectly by adding data or objects to an inbox (e.g., the EventCoord's InQueue, or the EventProcesser's InEvents).

Environment: CollabSteps and CollabSequences can be parameterized to vary the simulation behavior. Example properties are the publisher's event fire rate, the middleware's delivery delay, and the subscriber's number of topics.

Having defined the simulation structure and behavior, a major challenge still remains before executing and analyzing the simulation model. Complex, large-scale collaboration systems typically exhibit a considerable set of configuration parameters, such as connectivity, work duration, action probabilities, and simulation duration, which quickly leads to an explosion in possible simulation configurations. We propose two main mechanisms for reducing the configuration space (which we then exemplary demonstrate in the next subsection).

First, we suggest introducing dependencies between multiple configuration parameters. For example, all work durations can be defined as ratios of a core execution duration. This limits testing efforts to determination of sensible dependencies and limits simulation executions to a greatly reduced set of core parameters.

Second, we recommend the separate evaluation of independent patterns before aggregating them into the overall system. Doing so reveals the fundamental behavior and functional limits of a particular pattern, subsequently reducing the complexity of evaluating them as they are integrated into a composite system.

3.3 Scenario Model

Following the design methodology in the previous subsections, we provide one possible simulation model for the monitoring system from the scenario.

Structure: The motivating scenario introduced five user types: *sensors*, *field agents*, *back-office agents*, *back-office analysts*, and *team leaders* (recall Sec. 2). Fig. 5 depicts the interaction topology.[3] Field agents publishing information about the same topic also update a common status report using the shared artifact pattern. They thus adopt the publisher role in the pub-sub pattern and the contributor role in the shared artifact pattern. Similarly, back-office agents exhibit roles in three different patterns: publisher and subscriber, contributor, and master. Back-office analysts perform the worker role in the master-worker pattern. Collaboration connector tasks such as event distribution, artifact access control, and task assignment are implemented in software.

Output: In the building monitoring system, one global performance metric is the duration between when a critical situation occurs and when it is recognized

[3] A readable visualization of the overall simulation model would exceed this paper's page margins. View it at: `http://wp.me/P1xPeS-2F`

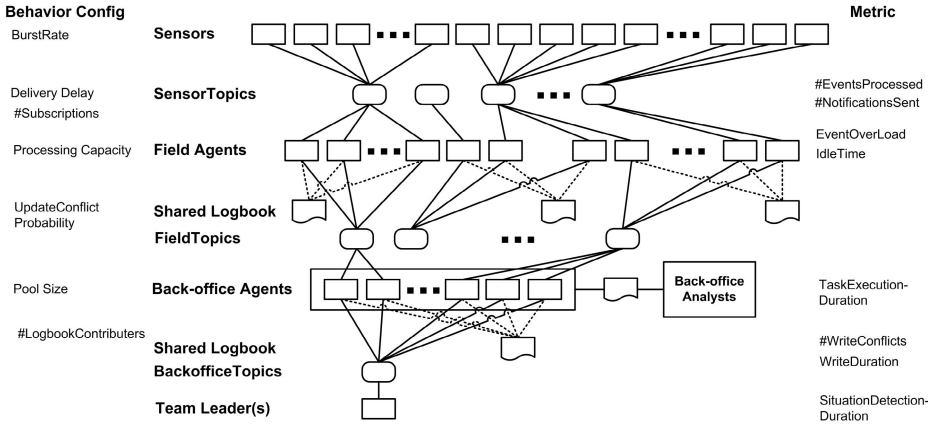

Fig. 5. Monitoring system simulation overview (simplified: collaboration connectors omitted for sake of clarity)

and responded to by the team leader. Suppose engineers are interested in the behavior under various load levels. The primary, external drivers of system load are the frequencies of sensor events, notification messages, analysis tasks, and log book updates. To make the simulation as realistic as possible, our model will include periods of regular, low-level activity interrupted by bursts of activity.

Using our approach, engineers can compare the effect of design decisions on system behavior, in terms of reaction to and recovery from increasing load levels. For example, we will show how engineers can examine the effect of (i) dropping events instead of processing all events, (ii) processing tasks in a last-in-first-out (LIFO) manner versus first-in-first-out (FIFO), and (iii) obtaining a write lock for the shared log book rather than updating on demand and resolving write conflicts later. Relevant metrics in the model capture agent load/idle time, task duration, time waiting to obtain a lock, and number of update conflicts. We discuss the system's emergent behavior and associated metrics in more detail for each pattern in the following subsections.

Logic: The behavior of individual agents and roles in the model is specified as follows. Within each collaborator, we separate activities associated with each role played by the collaborator to allow for individual pattern analysis (recall that some collaborators play multiple collaboration roles). Yet, creating such composite patterns requires integrating control and data flow within a single collaborator. Whereas the particular mechanism depends on the application, in general events will trigger processing while data passes through "inboxes" to the other pattern. As shown in Fig. 4, the Subscriber role enqueues all received PubMsgs to the EventProcesser's *InEvents* box and notifies the *FieldAgentWork* method about new events. The publisher component in turn will dispatch new PubMsg events from the EventProcesser via its *ProcEvents* box or otherwise idle and/or create a random new message on its own.

Environment: Modeling different design alternatives in separate models is not an option when aggregating multiple patterns, each having several configu-

ration options. Instead, we introduce parameters available for configuration at simulation time to switch between design variants. In the shared artifact pattern, for example, *DoArtifactLocking* determines whether CollabSequences to *GetLock*, *WaitLock*, and *ReleaseLock* methods (shaded) will be active or bypassed (Fig. 6).

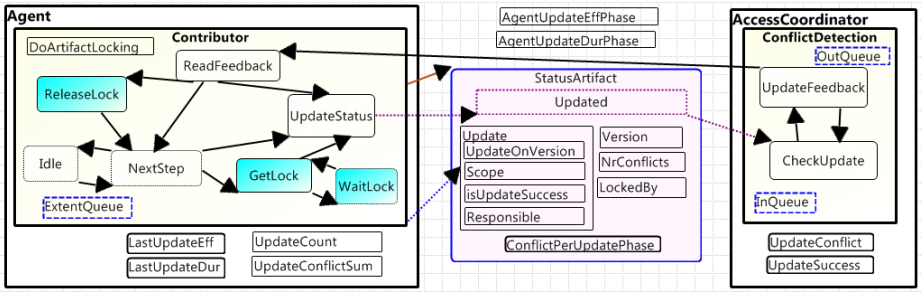

Fig. 6. Model excerpt for evaluating opportunistic write access and lock-based write access for updating a shared artifact

Following the recommendations in Section 3.2 on introducing configuration dependencies and thus minimizing the number of tuning parameters, all human execution methods derive their duration as a ratio of a core agent *WorkExecutionDuration* setting. Similarly, we defined the amount of topics and log books as a fixed rate of the number of sensors, respectively field agents.

4 Evaluation

The evaluation of our approach is two-fold. First, we show that modeling a complex large-scale collaboration system is indeed feasible (Sec. 4.1). Second, we demonstrate that our approach is beneficial during system architecture development. To this end, we analyze design decision trade-offs for individual collaboration patterns (Sec. 4.2) and for the composite pattern (Sec. 4.3) from the motivating scenario.

4.1 Feasibility

A feasible modeling approach should simultaneously facilitate simulations of small as well as complex models without involving considerable design overhead. We take the number of collaborators (*Coll*), components (*Comp*), connectors (*Conn*), collaboration steps (i.e., methods) (*CStep*), collaboration objects (*CObj*), collaboration sequences (*CSeq*), triggers (*Trig*), and trigger flows (*TFlow*) as an indicator for the modeling complexity. For the individual patterns we count only elements involved in the respective simulation run. Table 1 demonstrates that even a composite model needs only a few elements to model complex behavior. In our example, we minimized the number of collaborators by integrating all agent behavior in a single collaborator type.

Table 1. Model element count for individual and aggregated patterns. Note that each simulation contains elements for determining the active and calm event intervals.

Pattern	Coll	Comp	Conn	CStep	CObj	CSeq	Trig	TFlow
Publish/Subscribe	2	3	1	7	5	7	4	6
Master/Worker	3	2	1	7	3	7	4	6
Shared Artifact	2	1	1	9	2	14	5	6
Composite	5	6	3	25	8	36	9	15

4.2 Simulating Individual Patterns

Master/Worker Simulation: Suppose engineers wish to evaluate the effect of assigning tasks to back-office analysts in FIFO versus LIFO order. The master/-worker model measures the number of open tasks, the number of idle workers, and the task execution duration across time. We are interested in average task duration and also how predictable duration is (i.e., whether two sequential tasks tend to yield similar duration time).

Fig. 7 shows the data gathered from a simulation in which ten back-office agents (masters) each generate a task every time unit with 40% probability or otherwise idle. The pool of workers consists of twenty back-office analysts who require three time units (t) to complete a task. Task bursts are generated every $56t$ and last for $28t$. The simulation indicates whether the worker pool recovers from the added load in a timely manner (i.e, whether the recovery duration is less than the burst duration). During each burst, back-office agents double their task creation likelihood, and cut their idle time in half (resulting in a fourfold load increase). For each subsequent burst, the number of back-office agents in burst mode increases by 10%. Once 3000 tasks have been generated, no new tasks are created, allowing the analyst pool to finish all remaining tasks and obtain comparable metrics (FIFO and LIFO yield overall $15.1t$ average duration).[4]

Table 2. Average task duration (dur) and average sequential duration difference (diff) for FIFO and LIFO for the three phases of Fig.7.

		FIFO		LIFO		
	Interval (t)	dur	diff	dur	dur < 100	diff
Phase 1 Low	0-112	3.04	0.06	3.04	3.04	0.06
Phase 2 Med	113-280	7.57	0.25	10.21	6.92	6.51
Phase 3 High	281-450	26.14	0.29	24.04	12.44	13.82

FIFO and FIFO perform equally well for a low task load (Phase 1). Towards the end of Phase 2, the worker pool reaches its recovery limit, as it cannot process the load received during the burst phases rapidly enough. Note that LIFO

[4] Due to page constraints, we cannot display the very similar behavior observed for 1,000 agents, 2,000 analysts, and 300,000 tasks.

processing results in the first few tasks (duration $> 100t$) to be completed during simulation "cool down." Considering all tasks, FIFO appears to be superior to LIFO in this scenario. However, tasks that remain unprocessed for a very long time may become irrelevant. For example, if we ignore task as useless after $100t$, LIFO provides better average task duration for relevant responses. (Table 2: dur < 100). On the downside, LIFO causes large fluctuations in the durations of sequential tasks (Table 2 diff), regardless of load level. A task completed in minimum time might be followed by a task that potentially expires.

The simulation highlights various control parameters to steer the system behavior under load. Besides the main choice between FIFO and LIFO, engineers may decide to limit task creation rates, limit the number of back-office agents, increase the size of the analyst pool, or enforce task expiration dates. Evaluation of more sophisticated mechanisms such as task priorities, variations of worker performance, or dynamic switching between FIFO and LIFO merely requires some additional modeling work.

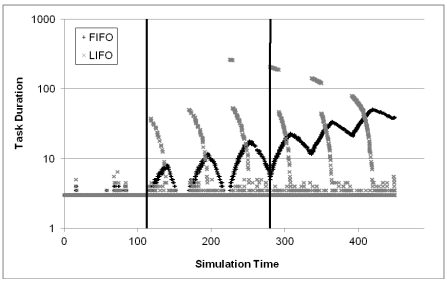

Fig. 7. Task execution duration for FIFO and LIFO with increasingly intense burst intervals

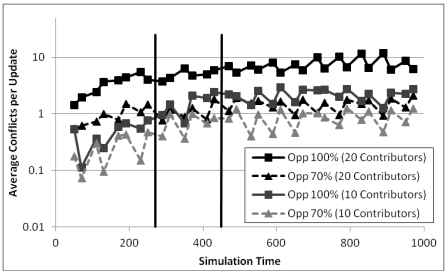

Fig. 8. Average number of conflicts until successful update

Shared Artifact Simulation: Use of a common logbook by multiple agents in the monitoring system raises the question of how to control write access. Engineers face a trade-off between lock-based access and opportunistic access with conflict detection. The former guarantees write success and constant write effort, but may cause long wait times to obtain the lock. The latter promises immediate write access at the chance of creating write conflicts, potentially imposing additional work to resolve conflicts. Relevant metrics in the shared artifact model (Fig. 6) capture the average duration to successfully complete an update and the average probability of a conflict occurring during an update.

In the shared artifact simulation, a set of agents (playing the role of contributors) access a single logbook. Periodically, each agent updates the logbook with 20% probability or idles (i.e., works on something else) for $6t$. An update attempt takes $1t$ and specifies the last known artifact version and the update scope. The scope ($[0; 100]$) describes the extent of the update (e.g., the portion of the logbook modified) and is equivalent to the likelihood of a conflict among different versions (0% = never conflicting, 100% = always conflicting). The system

behavior is simulated under fluctuating load, exhibiting bursts lasting $40t$ followed by periods of baseline behavior lasting $20t$. The remaining burst behavior configuration and generation is identical to the master/worker simulation above. Fig. 9 shows the update duration for twenty and ten contributors, respectively.

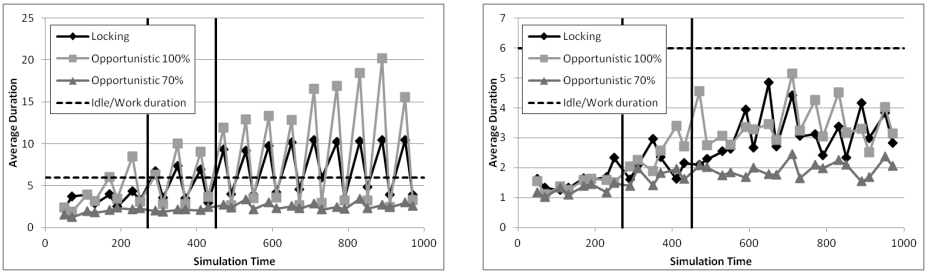

Fig. 9. Update duration for 20 (left) and 10 (right) contributors per artifact for lock-based and opportunistic access at 100% and 70% conflict likelihood

The two strategies perform similarly at low load as lock waiting times remain low and conflicts are infrequent. At medium load opportunistic updates show significant duration benefit for 70% conflict likelihood. Note that with conflict probability of 70% and 20 contributors, however, every update still fails on average more than once, which for humans is typically considered unacceptable (Fig. 8). At high load, neither strategy remains sensible. Locking for this configuration of contributors and write access duration hits its theoretical limit, in which agents are constantly waiting for a lock (i.e., average update duration of 10). Opportunistic access even yields average update durations well beyond twice the regular idle duration.

Engineers might consider some or all of the following options in this scenario: (i) (dynamically) restricting the number of contributors per artifact, (ii) enforcing update rate limits, and/or (iii) facilitating shorter access durations. Fig. 9 (right) highlights how reducing the contributor base (compared to Fig. 9 (left)) results in a disproportionately large reduction in duration and conflicts. Even then, with 10 contributors, 70% conflict probability produces one conflict per update on average during burst intervals (Fig.8).

4.3 Simulating Composite Patterns

Having gained an understanding of the master/worker and shared artifact patterns individually, we now focus on their aggregation, along with the pub/sub pattern, to assess the complex system described in Section 4.1. Specifically, our composite model simulates the effect of activity bursts on the analyst pool and log book to reveal the impact on the time required to detect unsafe or insecure situations. A typical simulation run for this scenario involves 600 collaborators, 100k events, 10k tasks, and 8k artifact updates within $1000t$.

Activity bursts in the composite model are identical to those in the previous simulations. Sensor events are tagged with a different situation ID for each $40t$ interval of base or burst behavior. We measure the duration from the beginning of an interval until the time the team leader first detects the ID from incoming events. The load on field agents determines the scope of logbook updates (i.e., the conflict likelihood) and the number of outgoing events. Those events result in tasks being assigned to back-office analysts for processing before being propagated to the team leader by back-office agents.

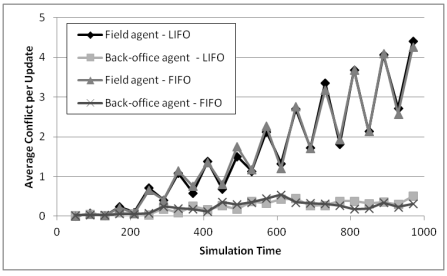

Fig. 10. Access conflicts for field agents and back-office agents

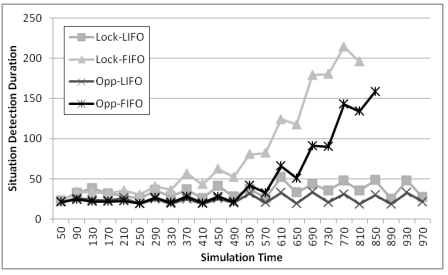

Fig. 11. Situation detection duration for lock-based and opportunistic access

Agents update the log-book after each iteration of event processing. Whereas lock-based access allows the agent to continue working while waiting, opportunistic access requires full attention and delays event processing while the update is occurring. Lock-based access requires the agent to complete each update before attempting a subsequent one.

In the composite model, field agents operate at the lock-based limit: 10 agents per artifact, updating about every $5t$, with update duration $1t$. As Fig. 11 highlights, the lock waiting time directly impacts timely situation detection. However, opportunistic access comes at the cost of rising conflicts for field agents with increasing event load. Back-office agents are less affected as only four share a single artifact and are additionally shielded from high load by previous field agent processing (Fig. 10).

Given the level of redundant events, LIFO clearly outperforms FIFO for medium and high load (Fig. 11). The increasing task load delays situation detection with FIFO to such extent, that the last few situations remain undetected within the simulation time.

In summary, the composite system simulation supports system design decisions by highlighting (i) the impact of access strategies on the detection duration, (ii) the impact of task queue style on situation detection success, (ii) the bottlenecks (i.e., shared artifacts for field agents rather than back-office agents), (iv) and the potential for dynamically switching between strategies. For example, applying FIFO and opportunistic access at low load and switching to LIFO and lock-based access at high load.

Limitations: A simulation can only give detailed recommendations about the optimum expert pool size, or the optimum number of experts per artifact for precisely defined models of individual patterns. Simulations of complex socio-technical systems can only cover particular aspects of interest, never all details. Thus any results in terms of absolute numbers are unsuitable to be applied directly in a real world systems. Instead, the simulation enables system engineers to compare the impact of different design decisions and decide what trade-offs need to be made. The simulation outcome provides an understanding what mechanisms might fail earlier, which strategies behave more predictably, and which configurations result in a more robust system. At the same time, a simulation raises awareness of system metrics that are best suited for serving as indicators of looming performance deterioration.

5 Related Work

Simulating system aspects for gaining an understanding of its behavior has been proposed in many diverse areas. Scenario-driven dynamic analysis of distributed architectures enables the system architect to compare design trade-offs [8]. Extensions to UML models such as sequence diagrams allows for tracking of performance metrics [2]. Simulation ranges from modeling individual software components [4] to large-scale service oriented systems [15] for the prediction of system reliability or the development of adequacy criteria and test cases for distributed systems [21]. Simulation of business process and workflows aims for detecting bottlenecks, predicting cost and time, evaluating quality and flexibility, and determining other performance metrics [14,23,17]. Research on crowdsourcing applies simulation to demonstrate the effect of assessment tasks on skill evolution [22], to evaluate the impact of collaboration policies [20], or determine the optimum number of replicated jobs per task [3].

These research efforts target important but only partial aspects of socio-technical systems. Focusing only on a subdomain or only on the technical part, they thus fall short of delivering collaboration-centric design support.

Social network analysis observes and analyzes general emerging system properties such as the power-law network topology [1] as well as system specific properties such as the structure of discussion threads on Slashdot [11]. Such research provides insights on how to simulate realistic structure and behavior of large-scale collaborative efforts.

On the small scale end of the spectrum, team automata formalize the interactions amongst multiple participants in groupware systems [9]. Although team automata initially targeted Computer Supported Cooperative Work (CSCW) systems to rigorously define and enforce collaboration protocols [16], their nature lend them more to the analysis and design of security mechanisms. We believe that team automata are unsuitable for simulating socio-technical systems where the exact participant behavior is a-priori unknown. The *Construct* group simulation tool [13] overcomes these limitations but remains severely restricted in the maximum amount of simultaneously active collaborators.

6 Conclusions

This paper presented a method for simulating complex collaboration structures in support of understanding system design trade-offs. We extended the human Architecture Description Language to obtain an executable model of collaboration patterns. Exemplary simulations of the master-worker pattern, the shared-artifact pattern, and their integration with the publish-subscribe pattern demonstrate feasibility and benefit to the system designer.

Future work will focus on exploring these patterns in more detail and applying the simulation framework for evaluating dynamic switching between strategies (e.g., $FIFO \Leftrightarrow LIFO$) at runtime and evaluate our work in the scope of a real world system. Simultaneously we intend to include additional human component aspects such as learning, forgetting, skills, trust, or social connections.

Acknowledgment. This work is supported in part by the Austrian Science Fund (FWF) under grant number J3068-N23.

References

1. Albert, R., Barabasi, A.L.: Statistical mechanics of complex networks. Reviews of Modern Physics 74, 47 (2002)
2. Balsamo, S., Marzolla, M.: Simulation modeling of uml software architectures. In: Proc. of ESM 2003, the 17th European Simulation Multiconference, pp. 562–567. SCS–European Publishing House (2003)
3. Brun, Y., Edwards, G., Young Bang, J., Medvidovic, N.: Smart redundancy for distributed computation. In: Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS 2011), pp. 665–676 (2011)
4. Cheung, L., Roshandel, R., Medvidovic, N., Golubchik, L.: Early prediction of software component reliability. In: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008, pp. 111–120. ACM, New York (2008)
5. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM 54, 86–96 (2011)
6. Dorn, C., Taylor, R.N.: Analyzing runtime adaptability of collaboration patterns. In: International Conference on Collaboration Technologies and Systems (CTS). IEEE Computer Society, Los Alamitos (2012)
7. Dorn, C., Taylor, R.N.: Architecture-Driven Modeling of Adaptive Collaboration Structures in Large-Scale Social Web Applications, Tech. Rep. UCI-ISR-12-5, University of California, Irvine (May 2012), `http://www.isr.uci.edu/tech_reports/UCI-ISR-12-5.pdf`
8. Edwards, G., Malek, S., Medvidovíc, N.: Scenario-Driven Dynamic Analysis of Distributed Architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 125–139. Springer, Heidelberg (2007)
9. Ellis, C.: Team automata for groupware systems. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge, GROUP 1997, pp. 415–424. ACM, New York (1997)
10. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)

11. Gómez, V., Kaltenbrunner, A., López, V.: Statistical analysis of the social network and discussion threads in slashdot. In: WWW 2008: Proceeding of the 17th International Conference on World Wide Web, pp. 645–654. ACM, NY (2008)
12. Gregg, D.G.: Designing for collective intelligence. Commun. ACM 53, 134–138 (2010)
13. Hirshman, B.R., Carley, K.M., Kowalchuck, M.J.: Specifying Agents in Construct. Tech. Rep. CMU-ISRI-07-107, Carnegie Mellon University (July 2007)
14. Hlupic, V., Robinson, S.: Business process modelling and analysis using discrete-event simulation. In: Proceedings of the 30th Conference on Winter Simulation, WSC 1998, pp. 1363–1370. IEEE Computer Society Press, CA (1998)
15. Juszczyk, L., Dustdar, S.: Script-based generation of dynamic testbeds for soa. In: Proceedings of the 2010 IEEE International Conference on Web Services, ICWS 2010, pp. 195–202. IEEE Computer Society, Washington, DC (2010)
16. Kleijn, J.: Team Automata for Cscw - a Survey. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 295–320. Springer, Heidelberg (2003)
17. Li, J., Fan, Y., Zhou, M.: Performance modeling and analysis of workflow. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 34(2), 229–242 (2004)
18. Malone, T.W., Laubacher, R., Dellarocas, C.: Harnessing crowds: Mapping the genome of collective intelligence. Technology 1(2), 327–335 (2010)
19. Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime software adaptation: framework, approaches, and styles. In: Companion of the 30th International Conference on Software Engineering, pp. 899–910. ACM, New York (2008)
20. Psaier, H., Skopik, F., Schall, D., Juszczyk, L., Treiber, M., Dustdar, S.: A programming model for self-adaptive open enterprise systems. In: Proceedings of the 5th International Workshop on Middleware for Service Oriented Computing, MW4SOC 2010, pp. 27–32. ACM, New York (2010)
21. Rutherford, M.J., Carzaniga, A., Wolf, A.L.: Evaluating test suites and adequacy criteria using simulation-based models of distributed systems. IEEE Trans. Softw. Eng. 34(4), 452–470 (2008)
22. Satzger, B., Psaier, H., Schall, D., Dustdar, S.: Stimulating Skill Evolution in Market-Based Crowdsourcing. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 66–82. Springer, Heidelberg (2011)
23. Tumay, K.: Business process simulation. In: Simulation Conference Proceedings, pp. 55–60 (December 1995)