Specifying Flexible Human Behavior in Interaction-Intensive Process Environments

Christoph Dorn[1], Schahram Dustdar[1], and Leon J. Osterweil[2]

[1] Distributed Systems Group, Vienna University of Technology
{dorn,dustdar}@dsg.tuwien.ac.at
[2] Department of Computer Science, University of Massachusetts Amherst
ljo@cs.umass.edu

Abstract. Fast changing business environments characterized by unpredictable variations call for flexible process-aware systems. The BPM community addressed this challenge through various approaches but little focus has been on how to specify (respectively constrain) flexible human involvement: how human process participants may collaborate on a task, how they may obtain a joint decision that drives the process, or how they may communicate out-of-band for clarifying task-vital information. Experience has shown that pure process languages are not necessarily the most appropriate technique for specifying such flexible behavior. Hence selecting appropriate modeling languages and strategies needs thorough investigation. To this end, this paper juxtaposes the capabilities of representative human-centric specification languages hADL and Little-JIL and demonstrate their joint applicability for modeling interaction-intensive processes.

1 Introduction

Over the past 15 years, process flexibility [15] has been consistently identified as a key aspect for addressing the challenges arising from fast changing business requirements and unpredictable runtime situations. Existing research approaches predominately address flexibility at the process, artifact, and resource level. Little focus has been given to flexible human involvement. Flexible human involvement gains particular importance in interaction-intensive environments. Typical processes occur frequently in the health-care domain or when jointly creating knowledge. These environments exhibit close collaboration among participants, ad-hoc communication, and dynamic decision making while maintaining regions of rigid control-flow constraints. Traditional approaches to process and workflow specification assume a single executing entity per task or activity. Any communication among participants remains implicit, respectively remains outside the specification's scope. CSCW and groupware approaches, on the other hand, offer extensive flexibility but lack sophisticated process support.

We propose a middle ground between these two "extreme" ends of the human interaction spectrum. Specifically, we suggest refining process tasks with human interaction patterns and vice versa. For example, a health-care process could specify that a particular flow-decision may be discussed in a chat room with the head-nurse as moderator. In the opposite direction, authors working on a joint report (i.e., a shared artifact pattern) may incorporate a process specifying steps to safeguard report quality, intellectual property protection, and data anonymity. Experience has shown that pure process

S. Sadiq, P. Soffer, and H. Völzer (Eds.): BPM 2014, LNCS 8659, pp. 366–373, 2014.

languages are not necessarily the most appropriate technique for specifying human interaction patterns. Imagine modeling collection, filtering, distribution, and floor control in a chat room merely in terms of task sequences. Utilizing only process modeling elements quickly becomes tedious, while from a process point of view only the decision outcome (made by the chat room participants) is ultimately of true relevance. Hence interaction-intensive processes require dedicated specification of user behavior beyond current process-centric approaches.

In this paper we consider two mechanisms: (a) the process-centric language *Little-JIL* [7], and (b) the structure-centric *human Architecture Description Language* (hADL) [9] (Sec. 3). Along these lines, we attempt to obtain better insights into jointly utilizing Little-JIL and hADL. Although there is strong support for using either language independently [14,10,9] our hypothesis is that applying both languages in combination will provide more intuitive results in interaction-intensive environments (Sec. 4).

2 Related Work

Work on process flexibility typically focus on adding, removing, replacing process fragments, extending loops, and reconfiguring control dependencies on the process type level and process instance level [16]. Schonenberg et al. [15] provide a taxonomy whereby they distinguish among flexibility *by design*, *by deviation*, *by underspecification*, and *by change*. FLOWer [1] and similar case-based approaches (often denoted as activity-based or ad-hoc work-flows [11]) excel at undoing, repeating, skipping, and including activities. Our work is highly complementary as we specifically focus on the process participants' collaboration flexibility which none of these approaches appropriately address. Further investigations into applying our approach to for example case management or business artifacts are highly appealing.

Recently, the BPM community started exploring the convergence of BPM technology and social media. Brambilla et al. present design patterns for integrating of social network features in BPMN [6]. A social network user may engage in task-centric actions such as voting, commenting, reading a message, or joining a task. Böhringer utilizes tagging, activity streams, and micro-blogging for merging ad-hoc activities into case management [5]. Dengler et al. utilize collaborative software such as Wikis and social networks for coordinating process activities [8]. At best, contemporary Social BPM approaches model collaboration as individual social network user actions; neither the actual collaborative activities nor their structure are explicitly specified. Our work, in contrast, focuses on modeling the actual collaboration among users (well beyond workflow coordinated tasks) in detail. We, thereby, treat the workflow and collaboration structure as equal models.

Also the software engineering community identified the need for combining process technology and collaboration support. Paulo Barthelmess provides an in-depth review of approaches to collaboration and coordination support [3]. Languages and tools primarily target coordination and collaboration via file-centric development artifacts and tasks. Serendipity [12] utilizes events, filters, and actions as the main coordination means among participants. SPADE [2] supports the integration and invocation of collaboration tools, but remains unaware what collaboration type and structure such external tools implement. Oz [4] builds upon a rule-based language for specifying which

users are allocated as task executors. Overall, modeling collaboration structures is crude and imprecise, often requiring tedious composition from low-level events. Any tightly integrated collaboration tools provide limited, fixed set of a/synchronous mechanisms that remain outside the process specification's modeling scope. Independent of research domain, we can claim that no process specification approach makes the distinction among collaboration connector and human component, nor provides dedicated decision support (e.g., voting) or information streams (e.g., subscriptions).

3 Specifying Human Flexible Behavior

The overall process specification needs to balance analyzability and flexibility. Orthogonal, the specification needs to differentiate between the three types of human involvement: communication, coordination, and work (co-)execution.

The **human Architecture Description Language** (hADL) [9] describes according to what structure humans interact to achieve a common (sub)goal such as discussing and subsequently jointly deciding upon resource usage (Fig. 2). hADL distinguishes between *HumanComponents* (light green rectangles) and *CollaborationConnectors* (dark green rectangles) to emphasize the difference between the primary collaborating users (e.g., a decision maker) and non-essential, replaceable users that coordinate the collaboration (e.g., a discussion moderator). A CollaborationConnector is thus responsible for the efficient and effective interaction among HumanComponents. Users typically employ diverse means of interaction that range from emails, to chat rooms, shared wiki pages, and Q&A forums, to vote collection. These means implement vastly different interaction semantics: a message is sent and received, a shared artifact can be edited, a vote can be cast. *CollaborationObjects* (rounded rectangles) abstract from concrete interaction tools and capture the semantic differences in subtypes; e.g., *Message* (yellow), *Stream* (light orange), or *SharedArtifact* (dark orange). *HumanActions* (tool icon) specify what capabilities a component or connector requires to fulfill his/her role, e.g., read a discussion thread or cast a vote. Complementary, a CollaborationObject signals its offered capabilities in the form of *ObjectActions* (gear-wheel icon). Both action types distinguish further between *Create*, *Read*, *Update*, and *Delete* (CRUD) privileges. Ultimately, *Links* connect ObjectActions and HumanActions to wire up HumanComponents, CollaborationConnectors, and CollaborationObjects into a collaboration structure. The *Pattern* provides a container for complex, hierarchical CollaborationObjects and interaction patterns composed from the elementary hADL elements. Element types, action CRUD privileges, as well as link cardinalities have no graphical representation and are edited as textual properties. The main motivation for hADL as a dedicated language is the separation between (i) CollaborationConnector and HumanComponent as well as (ii) the distinct CollaborationObjects. Languages such as UML are too vague to unambiguously model these differences. Even with extensions, they might tempt designers into modeling hADL aspects with non-hADL elements or contradict hADL's constraints and thus jeopardize rigorous analysis.

Little-JIL [7] is a visual language, depicting processes as hierarchies of steps (Fig. 1). An edge between a parent and child steps carries specifications of the arguments being passed between the two and an optional annotation specifying the number of child

step instances. Little-JIL incorporates four different step execution sequencing specifications: sequential (→), which specifies that substeps are to executed sequentially from left to right; parallel (=), which specifies fork-and-join for its substeps; choice (⊖), which specifies that only one of the step's substeps is to be executed, with the choice being made by the parent step; and try (⤫), which specifies that the step's substeps are to be executed in left-to-right order until one of them succeeds by failing to throw an exception. Exception handling is a particularly strong and important feature of Little-JIL. Exceptions may be thrown by a step's prerequisite check or postrequisite check or by the execution agent. Every step can contain one or more exception handlers, each of which may itself be an entire step hierarchy. A step's interface specification incorporates information about whether any arguments are an input, an output, or both, and the types of resources needed in order to perform the task associated with that step. One resource is always designated as the steps agent, namely the resource responsible for the performance of the step, may it be human(s), software, or hardware. This allows for linking Little-JIL steps to hADLs model elements, in a way that allows the two specifications to be orthogonal. Expressive, extensible, orthogonal resource specification and management [14] is thus one of the main reasons for choosing Little-JIL over workflow languages such as BPEL or YAWL. The way in which Little-JIL supports implementation of abstraction, based upon semantics rigorously defined using finite state machines, also facilitates clear specification of both activities and communication, as well as their relations to each other. These give the use of Little-JIL important advantages over other languages such as BPEL, BPMN, and YAWL.

Little-JIL vs. hADL

We analyze the spectrum between rigor and flexibility for multiple aspects as hADL and Little-JIL differ in their focus on where they enable precision, respectively underspecification.

Control flow describes the order relation among multiple actions, specifically interaction, coordination, and work execution steps. hADL assumes no single, dedicated control flow that determines the order of all human actions in a collaboration pattern. Instead, hADL enables specification of object lifecycle actions (CRUD) and who may trigger them. In contrast, Little-JIL offers primitives for rigorously determining the sequence and trigger conditions of steps.

Concurrency dependencies describe the active, simultaneous involvement of multiple users in the system. hADL assumes user behavior concurrent by default, only action sequences determined by an object's lifecycle imply order (i.e., first create, then read). Hence, actions such as multiple users reading a (shared) message or updating a shared artifact are expected to occur in no particular order with no synchronization mechanism involved. Little-JIL provides the "parallel step" primitive for marking a set of substeps explicitly as concurrently executable. Instantiating multiple identical substeps is achieved by annotation a step's edge with a fixed integer, a predicate, or a specification based upon the number of available resources.

Temporal, Cardinal, and Structural constraints provide additional refinement primitives that govern acceptable behaviors. hADL focuses primarily on minimal and maximal interaction cardinality, e.g., whether one or many users may update an artifact, the minimal number of reviewers of a report object, whether a user may create a single

or multiple task request objects. Little-JIL provides cardinality constraints for specifying the lower and upper bounds for repeatedly executing a particular step. Temporal constraints determine the maximum duration a step may take for completion. Resource constraints allow the precise selection of agents (filtered by properties), for example, expressing that the same agent must (or may not) execute a particular set of steps.

Communication primitives describe the various means and their properties for establishing unstructured communication among participants. hADL employs the CollaborationObject element (and its subtypes such as Message, Artifact, or Stream) for specifying the nature of communication and which communication role a particular user plays. CollaborationObjects thus may describe synchronous one-to-one video communication as well as asynchronous multiuser information exchange via a blackboard. Little-JIL facilitates communication among agents only via explicit data passing between steps.

Coordination primitives describe the available means for managing work dependencies among participants. hADL distinguishes between work-centric HumanComponents and coordination-centric CollaborationConnectors. Little-JIL relies on the process engine as the sole coordinator of human involvement (in contrast to hADL where multiple CollaborationConnectors are not uncommon). The process description serves as the sole coordination basis. Step output and resource availability determine the flow through the process, however from a human participant's point of view, there is no distinction between coordinating steps and work executing steps.

Execution primitives outline the basic language elements for specifying human behavior. In hADL, a HumanComponent's actions describes all capabilities required to fulfill the collaborative work task. A HumanComponent thereby makes use of actions made available by potentially multiple collaboration objects. Such an object may serve as work input/output, but also for coordination or communication. Little-JIL unambiguously specifies human work in the process' step definitions. Step definitions precisely define all required input data, and exactly what output is expected, respectively what exceptions may occur. Exception handling is a significant aspect of a Little-JIL process specification in contrast to hADL where exceptions have to be modeled as regular collaboration objects.

4 A Hospital Patient Handling Use Case

We evaluate to what degree the above outlined capabilities of hADL and Little-JIL manifest as synergies. We follow a simple strategy: a designer chooses the language(s) that support specifying the kind of details about which she would like to reason upon and understand. Where control is desired, she tends towards process specification in Little-JIL. Where flexibility and human initiative needs to be emphasized and understood, there she opts for hADL. In this use case, we analyze the potential for human flexibility in an exemplary emergency department (ED) process (see also [14]). Efficient and effective ED processes rely on optimal resource allocation. This includes determining the optimal number of personnel such as Physicians, Nurses, Triage Nurses, or Clerks, their activities, and constraints on the combination of activities and personnel. Typically, a hospital determines a-priori the various thresholds which remain unchanged during operation.

Whether a threshold is adequate, however, is highly dependent on the dynamic changing ED context and highlights the potential benefit for ad-hoc involving actual humans in the dynamic resource allocation decisions. We model the main ED process (Fig. 1) in Little-JIL and the flexible collaboration structures (Fig. 2) in hADL. The *EDProcessScope* step assumes registered patients need first placement in a bed. The *NurseOverloadHandler* becomes active when available nurses are unable to carry out *PutPatientInBed* and resource allocation rules yet keep triage nurses from substituting. In this situation, how triage nurses may volunteer upon coordination with the ED supervisor is left for specification in hADL. In case no triage nurse volunteers or the supervisor declines the substitution, the *AssignBedScope* step executes a blocking *PutPatientInBed* step. If a bed is unavailable, a nurse may initiate a swap or, upon failure, will simply wait for a bed blocking. After successful bed placement and subsequent *AssessAndTreatScope*, the *EvalLoad* step determines a switch from *FinalAssessmentSame* to *FinalAssessmentDiff* strategy or vice versa in order to maintain short patient Length-of-Stay (LOS). How the ad-hoc, collaborative decisions come about are generally outside Little-JIL's scope but rather specified in hADL. The integration among Little-JIL and hADL occurs through a step's executing agent; here *EvaluateLoadAgent* and *NurseOverloadHandler*. These agents may be human or software entities.

The hADL model (see Fig. 2) focuses on the coordination among *Physicians*, *Supervisor*, *TriageNurses*, and process steps agents when to switch assessment strategies, and when to volunteer for role substitution. The *NurseOverloadHandler* directs *NurseLoadAlerts* messages to the *TaskAllocator* connector. The connector in turn creates a *VolunteerSelection* artifact, invites *TriageNurses*, observes who indicates their availability, waits for *Supervisor* confirmation and only then returns a *ResourceSubstitutionConfig* message to the process. Similarly, the *StrategySwitcher* connector observes resource status and patient LOS, collects *Physicians'* opinions on whether to switch, considers a *Supervisor's* overruling, and notifies the *EvaluateLoadAgent* step instances asynchronously on the agreed *StrategyChange* via a message stream. The connector

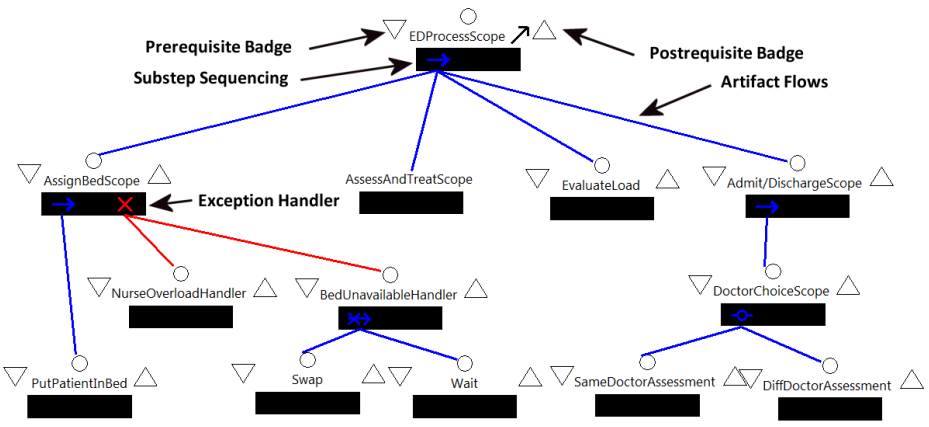

Fig. 1. Describing the adaptive ED process (excerpt) with Little-JIL

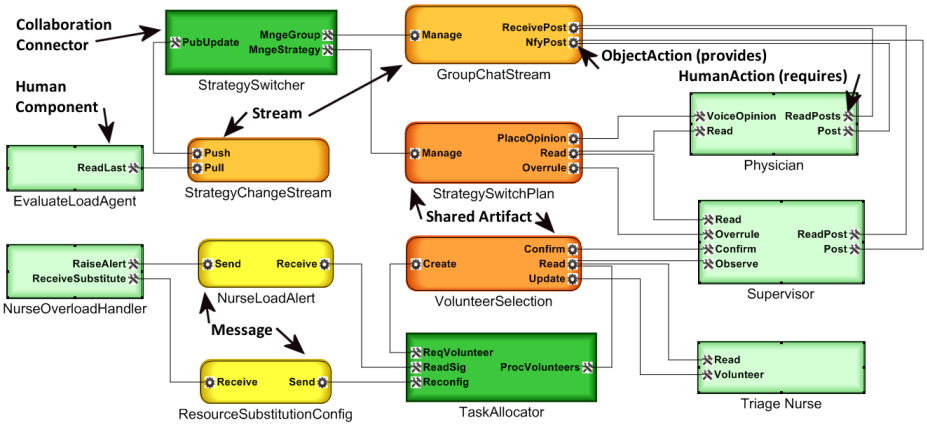

Fig. 2. Modeling collaboration structures in hADL

also manages a *GroupChatStream* that enables Physicians and Supervisor(s) to discuss strategy switching in an asynchronous and distributed manner. The sole purpose of the *TaskAllocator* connector and *StrategySwitcher* connector is coordinating collaboration among humans and integrating the collaboration with the process. Either connector might be implemented in software or by a dedicated user, e.g., a head nurse. The resulting decoupling ensures that process participants neither need to care about how to communicate with all relevant collaborators, nor do collaborators have to be physically collocated. Decoupling also provides the opportunity for establishing collaborations in parallel to multiple process instances.

Previous research [14,10,9] demonstrated the independent applicability of Little-JIL and hADL. Relying on a single language only, however, risks stretching it beyond its comfort zone. Modeling detailed processes in hADL quickly becomes tedious while still not completely achieving Little-JIL's rigor. Likewise, Little-JIL could describe the unstructured interaction occurring in a chat room but would need to do so at an extremely fine-grained level.

We believe that having to apprehend two languages neither puts an overly large nor unacceptable cognitive burden on the process designer. In software engineering, designers are equally expected to master the similarly diverse UML (or SysML) diagram types such as sequence diagrams vs. class diagrams.

5 Conclusions and Outlook

This paper[1] presented an approach for specifying human behavior in interaction-intensive process environments through the joint use of the Little-JIL and hADL. We outlined their differences as well as synergies and demonstrated their applicability in a

[1] This research was partially supported by the EU FP7 SmartSociety project (600854), the U.S. NSF under Award Nos. IIS-1239334 and CNS-1258588 and the NIST under grant 60NANB13D165.

use case. The combination of both languages avoids stretching one language beyond its comfort zone. Our approach will ultimately lead to more precisely specified human involvement and thereby enable better analysis of human actions as well as better support of their interaction needs. Our future investigation will focus on deployment issues such as instantiating collaboration patterns from a process engine and vice versa, observing collaborations, and detecting deviations from the initial model.

References

1. van der Aalst, W.M.P., Weske, M.: Case handling: A new paradigm for business process support. Data Knowl. Eng. 53(2), 129–162 (2005)
2. Bandinelli, S., Di Nitto, E., Fuggetta, A.: Supporting cooperation in the spade-1 environment. IEEE Trans. Softw. Eng. 22(12), 841–865 (1996)
3. Barthelmess, P.: Collaboration and coordination in process-centered software development environments: a review of the literature. Inf. and Soft. Tech. 45(13), 911–928 (2003)
4. Ben-Shaul, I., Skopp, P., Heineman, G., Tong, A., Popovich, S., Valetto, G.: Integrating groupware and process technologies in the oz environment. In: Proc. Int. Software Process Workshop, pp. 114–116 (October 1994)
5. Böhringer, M.: Emergent case management for ad-hoc processes: A solution based on microblogging and activity streams. In: zur Muehlen and Su [13], pp. 384–395
6. Brambilla, M., Fraternali, P., Vaca, C.: BPMN and design patterns for engineering social BPM solutions. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 219–230. Springer, Heidelberg (2012)
7. Cass, A.G., Lerner, B.S., Sutton Jr., S.M., McCall, E.K., Wise, A.E., Osterweil, L.J.: Little-jil/juliette: a process definition language and interpreter. In: ICSE, pp. 754–757. ACM (2000)
8. Dengler, F., Koschmider, A., Oberweis, A., Zhang, H.: Social software for coordination of collaborative process activities. In: zur Muehlen and Su [13], pp. 396–407
9. Dorn, C., Taylor, R.N.: Architecture-driven modeling of adaptive collaboration structures in large-scale social web applications. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 143–156. Springer, Heidelberg (2012)
10. Dorn, C., Taylor, R.N.: Coupling software architecture and human architecture for collaboration-aware system adaptation. In: ICSE, pp. 53–62. IEEE / ACM (2013)
11. Dustdar, S.: Caramba Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Vrtual Teams. Distributed Parallel Databases 15(1), 45–66 (2004)
12. Grundy, J., Hosking, J.: Serendipity: Integrated environment support for process modelling, enactment and work coordination. Automated Software Engineering 5(1), 27–60 (1998)
13. Jones, N.D., Muchnick, S.S.: Business Process Management Workshops - BPM 2010 International Workshops and Education Track, Revised Selected Papers. LNBIP, vol. 66. Springer, Heidelberg (1978)
14. Raunak, M.S., Osterweil, L.J.: Resource management for complex, dynamic environments. IEEE Trans. Software Eng. 39(3), 384–402 (2013)
15. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., Aalst, W.: Process flexibility: A survey of contemporary approaches. In: Dietz, J., Albani, A., Barjis, J. (eds.) Advances in Enterprise Engineering I, LNBIP, vol. 10, pp. 16–30. Springer Berlin Heidelberg (2008)
16. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)