A Novel Approach to Modeling Context-Aware and Social Collaboration Processes

Vitaliy Liptchinsky, Roman Khazankin,
Hong-Linh Truong, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology,
Argentinierstrasse 8/184-1, A-1040, Vienna, Austria
lastname@infosys.tuwien.ac.at
http://www.infosys.tuwien.ac.at

Abstract. Companies strive to retain the knowledge about their business processes by modeling them. However, non-routine people-intensive processes, such as distributed collaboration, are hard to model due to their unpredictable nature. Often such processes involve advanced activities, such as discovery of socially coherent teams or unbiased experts, or complex coordination towards reaching a consensus. Modeling such activities requires an expressive formal representation of process context, i.e. related actors and artifacts. Existing modeling approaches do not provide the necessary level of expressiveness to capture it. We therefore propose a novel modeling approach and a graphical notation, demonstrate their applicability and expressivity via several use cases, and discuss their strengths and weaknesses.

Keywords: Process Modeling, Social Context, Collaboration, Visual Language.

1 Introduction

Companies strive to retain the knowledge about their business processes by modeling them. If captured accurately, such knowledge allows us to analyze, improve, and execute those processes with higher efficiency. Although a variety of techniques and tools have been introduced for Business Process Modeling (BPM), nevertheless, modeling of highly dynamic non-routine processes such as human collaboration is still a subject for discussion in research [16].

While collaboration in general means working together to achieve a goal [8], the more narrow notion of creative human collaboration implies working together to design or improve some artifact (a piece of software, a wiki page, a product design, an article of law, a research paper, etc.). With proliferation of collaboration software, such as groupware or wikis, the manner of such collaboration has taken the form of incremental contributions to a network of shared documents. Relations between documents, actors, and other artifacts may influence the collaboration process. For example, some tasks should be done by actors chosen based on social relations, actions on some documents should not be performed

J. Ralyté et al. (Eds.): CAiSE 2012, LNCS 7328, pp. 565–580, 2012.

before related documents reach certain condition, or a change in a related document might force to re-do an activity. Although artifact-based process models have already been researched [19,2,4], existing modeling approaches do not emphasize the relations between artifacts as process "driving force", and, therefore, either do not provide the needed expressivity to capture this logic, or are inefficient because of the excessive complexity. We thus propose a novel modeling approach and a graphical notation for collaboration processes, the key principle of which is to treat each document's evolution as an individual process which is explicitly influenced by the states of related documents and patterns in surrounding social network. We propose to formalize the relations in line with the data from collaboration software, e.g., two developers can be considered related if they committed code to the same project folder in a source code repository. The amount of such data will grow with social computing pervading the enterprise IT[1], thus allowing process modelers to create richer models of people-intensive processes.

The rest of this paper is organized as follows: Section 2 describes motivation behind the modeling approach and presents a motivating example. In Section 3 we show the lack of expressivity in existing modeling approaches with regards to the motivating example. Section 4 describes the proposed modeling paradigm and the corresponding graphical notation. Section 5 demonstrates the usability of the approach through sensible use-cases. Disadvantages of the modeling approach are discussed in Section 6. The paper is concluded in Section 7.

2 Motivation

Collaboration is a recursive [13] process comprised of human interactions towards realization of shared goals. Groupware and social software foster collaboration of individuals who work across time, space, cultural and organizational boundaries, i.e., virtual teams [17]. Using this type of software, people interact through conversations (e.g., e-mails and instant messages) and transactions (e.g., create/modify/assign/restructure a document) in order to augment a common deliverable, e.g., documentation of an idea, a technical specification, a source code file, or a wiki page. Typically, such interactions are chaotic, non-routine, and are hard to predict and model. However, as side-effects they produce semantical and social relations between actors and artifacts. Furthermore, artifacts usually are semantically connected into hierarchical or network structures, i.e., references in wiki pages, or dependencies between software components.

As a motivating example, let us consider in-house software engineering in a dot-com company. Projects, or ventures, in such company can be classified as engineering ventures (development of new functionality), or analysis ventures (incident investigation, proof-of-concepts). Both types of ventures produce deliverables, such as source code or technical documentation. Figure 1 demonstrates a snapshot of a collaboration process as a directed graph of venture deliverables

[1] `http://www.gartner.com/it/page.jsp?id=1470115`

and collaborating actors. Edges connecting Ventures represent functional dependencies (i.e., venture depends on either an investigation report or a software component produced by other ventures). Edges connecting Actors depict social relations, i.e., there is a regular communication over instant messaging channels between them, or they contribute to the same venture. Analysis ventures, representing rather creative and non-routine work, can reside only in two possible phases, namely `In Progress` and `Finished`, while engineering ventures, representing more structured and long-running work, can reside in more phases, such as `Design`, `Implementation`, `Testing`, and `Finished`.

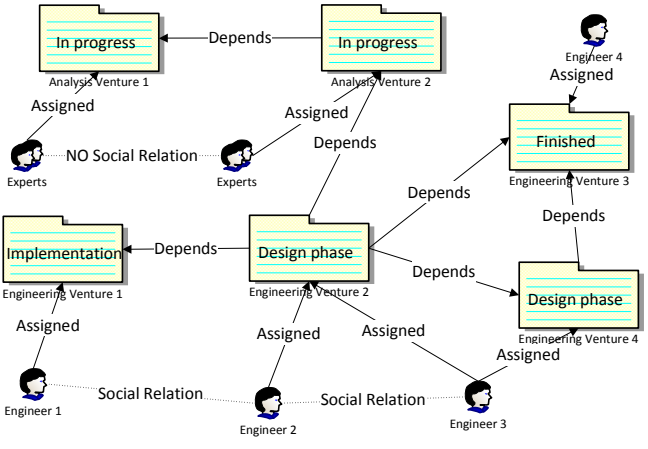

Fig. 1. Software engineering collaboration process snapshot

Now, let us consider a process modeler that possesses knowledge of working environment, culture, and the scale of the company, and aims at modeling the following rules:

1. *A venture project team should be notified of any changes in the technical documentation of other ventures it depends on. However, if two functionally interdependent ventures share any team members, then enforced communication is not required.* This rule ensures proper knowledge sharing between functionally interdependent ventures while avoiding overcommunication. For example, any new technical reports of `Analysis Venture 2` should be communicated to the project team of `Engineering Venture 2`. However, the same synchronization between `Engineering Venture 2` and `Engineering Venture 4` is not critical, because `Engineer 3` is anyway aware of any such changes.
2. *Venture technical documentation (i.e., design, or a report) should be reviewed by an expert from a functionally dependent venture. Moreover, it is preferable to assign an expert socially unrelated to the venture team members.* This rule tries to avoid biased reviews by finding a socially unrelated experts. For example, it is more preferable to assign `Engineer 4`, than `Engineer 1`, as a

reviewer of `Engineering Venture 2`, as `Engineer 4` does not have strong social relations with the `Engineering Venture 2` team.

3. *An engineering venture can be started if at least one venture, it depends on, has passed* `Design phase`. This rule defines a balance between total serialization of dependent ventures `Design phases`, which results in a longer time-to-market, and total parallelization of `Design phases`, which results in more iterations. For example, `Engineering Venture 2` was started upon completion of `Design phase` of either `Engineering Venture 3` or `Engineering Venture 1`.
4. `Design phase` *of a venture cannot be finished if all ventures, it depends on, have not passed* `Design phase`. This rule minimizes chances of potential rework and wasted efforts. For example, `Design phase` of `Engineering Venture 2` can be finished only after `Engineering Venture 4` switches to `Implementation phase`.

We refer to such rules as context dependency rules (CDRs). As it can be seen from the examples above, they allow to capture the knowledge about the impact of social and structural relations on collaboration processes. Formal specification can help visualize and improve CDRs, thus reflecting management experience in enterprise.

3 Related Work

In this section we discuss the related works and show their shortcomings with regard to their ability to model context dependency rules (CDRs), examples of which are outlined in the previous section. To the best of our knowledge, no framework is capable of capturing CDRs in a formal and visual manner.

Information-centric modeling approaches, such as Case Handling [2] and Artifact-centric workflows [4], can capture the evolvement of collaboration entities into formal models, and capture the relations on a conceptual level using composite cases and 'is-a' relationships between Roles in Case Handling, and Entity-Relationship models in Artifact-centric workflows. However, *condition* elements in these approaches do not allow to specify CDRs. Conditions in case-handling are defined as sets of bindings where a binding is a set of values for specific data objects. Therefore, it is not possible to define a condition which examines all the objects in a specific relation to the object at hand (CDR example 3), or to specify that *all* the related objects must reside in a specific state (CDR example 4). Conditions in Artifact-centric workflows are specified in formulas written in first-order logic. However, the specification is restricted and does not allow to use quantifiers, which is crucial for expressing CDRs (e.g., CDR examples 3 or 4).

Traditional activity-oriented business process modeling approaches like BPMN[2] allow to model dependencies between processes via messages or events. Asynchronous messaging can be used to partially resemble CDRs, e.g., by sending notifications to related processes. However, it would not provide enough flexibility to capture such rules. Using external events is another way to model

[2] `http://www.bpmn.org/`

such logic, but, it would require the specification of events in natural language. Moreover, activity-oriented approaches are difficult to apply for collaboration processes, because it is hard to predefine exact steps to follow [16]. In addition, explicit communication and coordination entities (i.e., events, message channels), intended for publishing information, do not convey any functional load and, therefore, complicate and encumber process models. Agent-based or agent-inspired approaches for coordination of business processes as [1,11] also utilize explicit information publishing entities, thus sharing the same disadvantages.

Context-aware workflows [20] are a generic approach that advocates the augmentation of workflow technology with information about the physical world. It is an execution framework, and proposes to use an XML-based language to express context dependencies. Theoretically, CDRs could be implemented using this framework, however, it would be hardly intuitive to comprehend.

In [3] so-called *batch-tasks* were proposed to allow for a task that is executed for multiple workflow instances at the same time. Other similar approaches can be found in [5]. Partially, CDRs can be covered by batch-tasks, e.g., CDR example 4. For more complex rules, however, this approach is not flexible enough.

Team Automata [9,10] use communication via shared action spaces. Transitions, which include the same external action, are fired simultaneously in these Automata. Alike to batch-tasks, it doesn't provide the needed flexibility.

COREPRO modeling framework [14] proposes to model the dependencies between states of related processes via so-called external state transitions. Again, it provides limited expressivity for describing the dependencies, as it allows to specify only exact external state transitions.

Futhermore, neither of approaches discussed above focuses on functional or social relations between actors and artifacts and therefore does not provide corresponding modeling elements. This makes it difficult for a modeler to specify such relations and their impact in a natural way.

4 Modeling Paradigm

In this section we present the modeling approach for collaboration processes, which allows to express context dependency rules (CDRs, see Sec. 2). We explain the design features, outline the modeling paradigm, and present modeling elements and graphical notation.

The key modeling principles of our modeling paradigm are:

- *Information-centric.* As described in Sec. 2, collaboration can be seen as a network of evolving artifacts. In addition, activity-oriented approaches are difficult to apply to collaboration processes, because it is hard to predefine exact steps to follow [16]. For instance, people interactions, such as conversations and transactions, in a collaboration process are rather chaotic and unpredictable, therefore, it is easier to capture collaboration artifacts and corresponding social and semantic relations as side effects of interactions. Therefore, the information-centric modeling paradigm is chosen as a basis for the modeling approach.

- *Bottom-up and neighborhood-aware.* Modeling an evolvement of a network of artifacts and people in a holistic view can be a daunting task. Contrarily, neglecting relations completely and modeling the progress of artifacts in isolation leads to context tunneling, and, therefore ineffective models. We thus propose to use a bottom-up hybrid approach, which models evolution of each artifact as an individual process explicitly influenced by its neighborhood. This approach allows to describe behavior at macro level (network of artifacts) by means of modeling behaviors at micro level (evolvement of a single artifact). Additionally, it allows to provide simple processes coordination and secure encapsulation: a process can modify only its own state, it cannot impact related processes explicitly. This approach was inspired by a computational model of Cellular Automata(CA) [15].
- *Social.* Collaboration processes often involve non-routine activities, such as discovery of socially coherent teams, or complex decision-making by exploiting social hubs and unbiased experts. Therefore, the paradigm promotes modeling not only a network of evolving artifacts, but also an evolving network of people.

In the following subsection we introduce the modeling framework, which incorporates the key principles discussed above.

4.1 Modeling Framework

Our modeling framework is defined as a set of basic modeling elements that a business process modeler can operate with in order to reflect context dependency rules (CDRs) within business process models:

1. *Collaboration artifacts* and their *states.* Artifacts should represent various aspects and deliverables of collaboration process (e.g., a software component, or a technical design). The states should represent the possible phases of collaboration. *Artifacts* and their *states* may be modeled using existing information-centric approaches, such as Artifact-centric workflows [4], making thus our approach rather complementory, than stand-alone.
2. *Relations.* Relations can be pre-defined (e.g., functional or structural dependency) or dynamic (e.g., temporal or social relations), i.e., produced as side effects of interactions and transactions. Proliferation of groupware and social software boosts the quantity and quality of dynamic relations data, thus empowering process modelers.
3. *Context-aware state transitions.* Context-aware state transitions define what *Relations* and *Artifacts* are relevant for a business process at various steps of its execution.

In order to better demonstrate how the framework basic modeling elements can be put together to model a business process, we present a graphical notation for the modeling framework. The notation is an extension of the conventional statecharts visual formalism [7]. The choice of statecharts is justified by their information-centric nature and widespread adoption as part of Unified Modeling

Language (UML)[3]. Being a natural visual representation of state machine mathematical model, statecharts include the following basic elements: (i) Clustered and refined *states*; (ii) State *transitions* comprised of *events* (external happenings such as user input or timeout), *conditions* (boolean expressions over events and state) and *actions* (e.g., sending an e-mail, or assigning a person to a task).

Our graphical notation, dealing with explicit modeling of relations, extends conventional statecharts with a new element `Context`, graphically depicted as a hexagon. `Context` element, being inseparable to `State` element, defines relations and artifacts, relevant to a particular state. Each `Context` element contains a query against the neighborhood of the artifact (i.e., related artifacts and people) asking for the presence of a specific pattern. Each `Context` can have several `Transition` elements attached: if the context query finds the corresponding pattern, then all transitions attached to this `Context` element are enabled, otherwise disabled. Similarly to `State` elements in statecharts, `Context` elements can be clustered using logical AND/OR/XOR operations.

Figure 2 demonstrates the overall integration of `Context` element into statecharts (the context queries are omitted in this figure for the sake of simplicity). Two of three transitions in the figure are enabled by `Context` elements. By default, we assume that transitions attached to `Context` elements have a higher priority over other transitions, but generally it is up to a modeler to define the priorities. Below are enlisted possible transitions in the default prioritization order:

1. If a pattern described in `Context 2` is found, then the state machine switches to state `D`. Here we can see that an *event* element is optional, and if absent, then the *transition* is activated at once.
2. If `Event 1` is fired and a pattern described in `Context 1` is found, then the state machine switches to state `B`.
3. If `Event 1` is fired and a pattern described in `Context 1` is not found, then the state machine switches to state `C`.

When modeling the behavior of multiple interdependent concurrent process instances, a modeler should assume that state transitions are synchronized, i.e., *every* Context element is evaluated before activation of *any* state transition in any process. Thus, if some process switches to state `A` and then instantly to some other state, the fact that it has been in state `A` will be considered.

We believe that graphs *a priori* are rather a natural visual medium for describing artifact networks and relations. Therefore, we define a visual graph query language, which is used to specify queries in `Context` elements. Queries expressed in the language can easily be mapped to a *First-Order Logic* expressions, but vice versa does not hold. A query in the visual language is a directed connected multigraph with labeled edges and nodes. Labels can either denote atomic relations/states/types, or expressions over atomic entities based on propositional calculus expressions. Additionally, labels may be absent in general, denoting a placeholder (e.g., any relation/state/type). An edge direction in a graph is used to depict a non-commutative relation.

[3] `http://www.omg.org/spec/UML/`

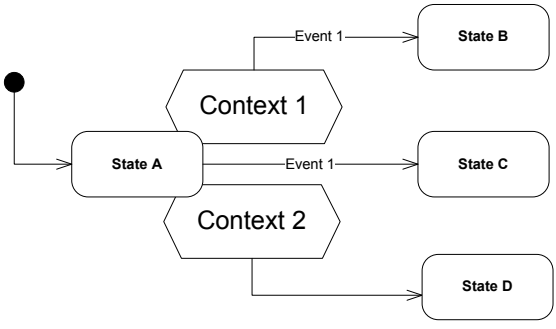

Fig. 2. Integration of `Context` elements into statecharts

Interpretation of a graph query naturally corresponds to the way we read *First-Order Logic* expressions. Query graphs always have one initialized primary element, therefore, graph queries should be interpreted outwards: starting from the central primary element towards most distant nodes. For example, graph queries depicted in Fig. 3 can be interpreted as follows:

- `Context 1`: if the primary document is in state `A`, and there are no documents, related by content or author to the primary one, residing either in state `A` or state `B`, then the attached transition is enabled.
- `Context 2`: if the primary document is in state `A`, and every single document, related by content to the primary one, must reside in state `B` and have two socially unrelated Authors that contributed to it, one of which is `Active`, then the attached transition is enabled.

As depicted in Fig. 3, single line edges correspond to existence quantifiers, while double line and crossed dashed edges correspond to universal quantifiers. Nodes in query graphs may be labeled with variables, that can later be reused in Conditions and Activities of corresponding Transitions. Since multiple occurrences of a context pattern may be found in the neighborhood, Activities/Conditions may be also extended with quantifiers, i.e., send e-mail to any/every related contributor.

The success of a modeling approach depends, to a great extent, on the level of simplicity offered. Therefore, we favor simplicity over completeness and impose following constraints on the queries expressed in the visual language:

- Only basic operators from proposition calculus are allowed as literal expressions attached to edges and nodes: conjunction, disjunction and negation. Even though, conditional and biconditional operators may be expressed via the former ones, more complex operators may decrease understanding and make reasoning about the model more difficult.
- Under Open World Assumption [18] negation may introduce ambiguity, therefore only negation as a failure is allowed, i.e., negation on an edge can be used only if nodes connected by the edge are transitively connected to the central node with non-negative edges.

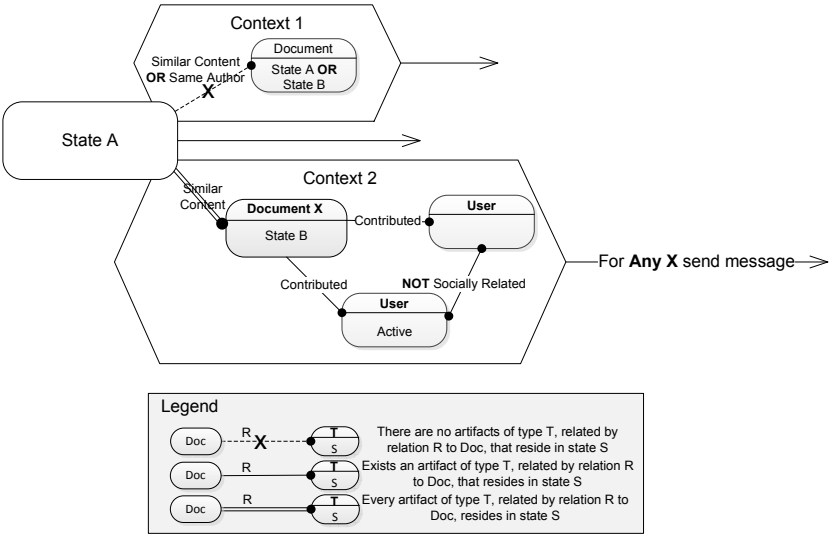

Fig. 3. Example of context queries in `Context` elements

– Edges with universal quantification can be adjacent only to the primary node. During our experiments with the modeling notation we observed that universal quantification can introduce ambiguity and would require assigning priorities to edges thus unnecessary complicating the modeling process. Nevertheless, implicit prioritization is still necessary: edges with universal quantification should have implicitly higher priority, than edges with existential quantification, in order to avoid ambiguities in case of cyclic query graphs.

A formal definition of our modeling notation is given below. In order to keep the definition succinct, we omit a formal definition of Statecharts, as it is available elsewhere, e.g., in [12].

Definition 1. Labels L in a query graph representing relations R, types T and states S of artifacts are defined as:

$$\begin{aligned} &\mathit{Label}\ L \stackrel{\mathbf{def}}{=} \mathit{Atomic\ Condition} \mid \mathit{Placeholder} \mid L\ \wedge\ L \mid L\ \vee\ L \mid \neg L, \\ &\mathit{Placeholder}\ \text{denotes}\ \mathbf{any}\ \text{value (no condition)} \end{aligned} \tag{1}$$

Definition 2. Edges in a query graph, along with adjacent vertices, are interpreted in First-Order Logic as follows:

$$(A) \xrightarrow{R} \bullet\ (T,S) \stackrel{\mathbf{def}}{=} \exists X : R(A,X) \wedge T(X) \wedge S(X) \tag{2}$$

$$(A) \xRightarrow{R} \bullet\ (T,S) \stackrel{\mathbf{def}}{=} \forall X : R(A,X) \wedge T(X) \rightarrow S(X) \tag{3}$$

$$(A) \xrightarrow{R} \times \bullet\ (T,S) \stackrel{\mathbf{def}}{=} \nexists X : R(A,X) \wedge T(X) \wedge S(X) \tag{4}$$

Where, given that graph queries are interpreted outwards from the central primary element (vertex), A denotes an already interpreted vertex. Predicates T and S describe type and state of suitable artifacts respectively. Result of query graph interpretation is a logical conjunction of the First-Order Logic formulas corresponding to graph edges. Higher priority of edges with universal quantification ensure that the corresponding to these edges formulas always appear at the beginning of the resulting logical conjunction.

Definition 3. Query graph Q is a quadruple defined as follows:

$$\begin{aligned} Q \stackrel{\textbf{def}}{=} (a, CE, V, EV), &\text{graph } Q \text{ is connected,} \\ &a \text{ is the predefined central primary vertex (artifact),} \\ &V \text{ is a set of vertices } (T, S), a \notin V, \\ &CE \text{ is a set of edges } CE \subseteq \{a\} \times \{\multimap, =\!\!\bullet, \text{-}\times\!\bullet\} \times V, \\ &EV \text{ is a set of edges } EV \subseteq V \times \{\multimap\} \times V \end{aligned} \tag{5}$$

Definition 4. `Context` element CTX in the modeling notation is a composition of query graphs CQ:

$$\begin{aligned} CQ \stackrel{\textbf{def}}{=} Q \mid & CQ' \textbf{ AND } CQ'' \mid CQ' \textbf{ OR } CQ'' \mid CQ' \textbf{ XOR } CQ'', \\ & CQ' = (a', CE', V', EV'), CQ'' = (a'', CE'', V'', EV''), \\ & a' = a'', CE' \cap CE'' = \emptyset, V' \cap V'' = \emptyset, EV' \cap EV'' = \emptyset \end{aligned} \tag{6}$$

Definition 5. Transition element CT, attached to Context element CTX, can be defined as:

$$\begin{aligned} CT \stackrel{\textbf{def}}{=} (CTX, E, C, AC), &\\ &E \text{ is an external event,} \\ &C \text{ is a condition, } C : QU \times ID \to \{\textbf{true}, \textbf{false}\}, \\ &AC \text{ is an activity, } AC : QU \times ID \to \emptyset, \\ &ID \text{ is a set of identifiers attached to vertices in } CTX \text{ graph,} \\ &QU \text{ is a set of quantifiers, } QU = \{\textbf{Any}, \textbf{Every}, \textbf{All}\} \end{aligned} \tag{7}$$

Our visual graph querying language was inspired by Graphlog language [6]. Graphlog is more complex, because it was designed as an execution language, as opposed to our language which aims rather at modeling. Our language assumes that central artifact is always present and exploits that to simplify universal quantification notation with special types of edges, while in Graphlog universal quantification is represented by a conjunction of existential quantification and negation.

5 Use Cases

This section describes three collaboration process use-cases which demonstrate the application of our modeling approach to various collaboration issues. As it

can be witnessed, the approach allows to easily express the dependency of a process on complex relations in its environment, and to compactly capture the dynamic co-influence between instances of the same process in one model. For clarity, in the use cases we attach to each `Context` element a free text description of its query.

5.1 Use Case - Design Game

Goal. The goal in this use case is to coordinate a design of a complex system consisting of interrelated projects. A set of expert virtual teams thus collaborate to reach a consensus. Assignment relation between teams and projects is one-to-one, but teams can share members. As some projects are dependent, it can happen that changes in the design of one project can be the reason for changes in the design of others. Finally, all project designs should be consistent with dependent ones.

Model. Each project of this system is regarded as a separate process (See Figure 4). In the beginning, it is in `In Progress` state which means the team is currently working on its design. When the team makes some changes to the design and commits it, the process goes into `Updated` state. If no changes to the design were made, i.e., the existing version was examined and considered valid, then the process switches to `Finalized` state. These two states represent superstate `Wait Input` which means that the project design is currently awaiting for some external actions. If the team suddenly decides to update the design (e.g., a better idea emerged), the process goes back into `In Progress` state.

Now, if the process is in `Wait Input` state, and if all the related projects are also in `Wait input` state and at least one is `Updated`, then the team should check the design of their project against inconsistencies with updated projects. Thus, the updated documents are sent to the team and the state is switched to `In progress`. An exception is the case when the project team shares a common expert with the team of an updated project(relation *Socially related*), who is expected to foresee any inconsistencies beforehand. Waiting the related projects

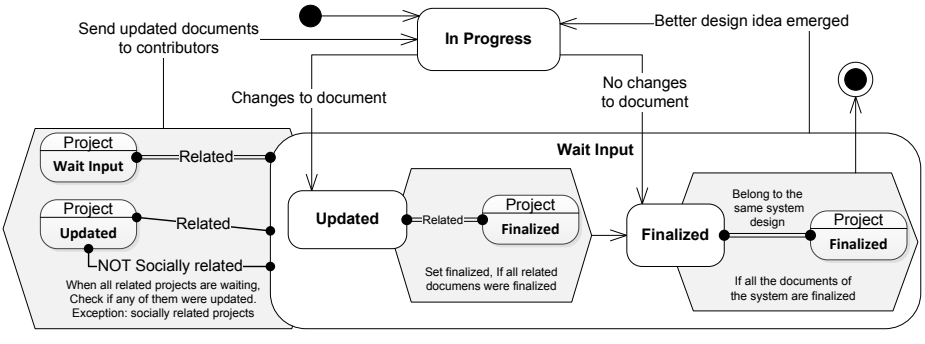

Fig. 4. Use case - design game

to be in "Wait input" ensures that all the updates of related documents will be taken into account.

When in `Updated` state, and if all the related projects are finalized, the process goes into finalized state, which ensures that if a document spawned no updates among related documents, it will not stay in `Updated` state.

The system may be considered in the final state when all the projects are in `Finalized` state.

Advantages. This use case demonstrates the modeling of collaboration as ordered iterative communication of project teams towards reaching a consensus. It shows that our modeling approach, as opposed to existing modeling approaches (See Sec. 3), is capable of expressing universal and existential quantification.

5.2 Use Case - Social Selection

Goal. The goal of this use case is to support a software development process with the selection of appropriate actors (e.g., developer, adviser, reviewer) based on relations with the other tasks and among the actors. Tasks are related if they belong to the same project, employees are related if they collaborated before.

Model. Figure 5 depicts the software development process. At first, the task is `Ready for implementation` state and is waiting for an appropriate developer to be assigned. Any available developer from a related task is assigned for this role, as he/she expected to be more productive because of being familiar with some related concepts. Alternatively, a manual assignment is performed. In either case, the process goes to `Implementation in Progress` state. An impediment can occur during the implementation (`Impediment pending` state), in which case an

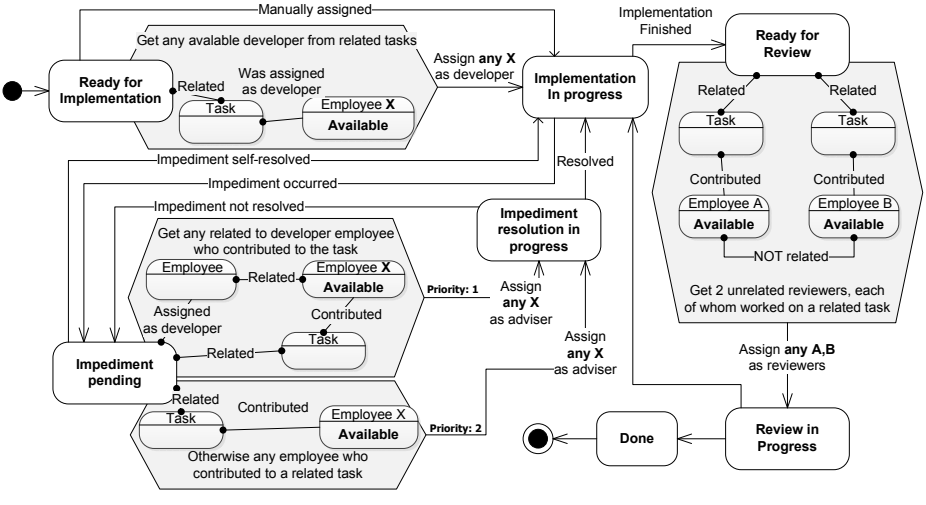

Fig. 5. Use case - social selection

adviser is needed for assistance. An adviser is preferably selected as a related to the developer employee who contributed to a related task, because of joint work experience. Otherwise, any related task contributor is chosen. If the adviser is found, the process goes into `Resolution in Progress` state, from where it can either go either back to `Implementation in Progress` or `Impediment pending` states, depending on whether the impediment has been resolved. Also, the developer can resolve the impediment by him/herself if no adviser was found. After the implementation is finished, the reviewers are selected (`Ready For Review` state): they are desired to have experience with related tasks but be unrelated to each other, which assures unbiased reviews. After the review process (`Review In Progress` state), either the implementation needs to be revised, or the task is considered finished.

Advantages. This use case demonstrates expressiveness of the modeling approach when visualizing social network environment, allowing thus to model processes that require discovery (e.g., compose a socially coherent team), unbiasedness (e.g., involve independent people), and negotiation (e.g. by exploiting of social hubs). It shows expresiveness of the graphical notation with regards to modeling patterns in a surrounding social network. Contrarily, existing modeling approaches do not model social relations between actors, therefore, are not capable of capturing such patterns (See Sec. 3).

5.3 Use Case - Dependent Components

Goal. The goal is to coordinate the development and testing of a software product, which consists of manifold components, some of which depend on others (we assume no cyclic dependencies). The development a component should proceed only when the components it depends on have reached certain progress.

Model. Figure 6 depicts the process which corresponds to a single component. It starts in `Open` state and switches over to `Implementation Phase` in either of two cases: it does not depend on any components, or at least one component which it depends on is in `Testing Phase`. This ensures some minimal basis for the development. After `Implementation phase`, the component is ready to switch over to `Testing Phase`, but, first, it should wait for all the components it depends on to be implemented, so the testing covers the combined functionality. The testing phase can reveal some flaws so the component will return into `Implementation Phase` for fixing those. If, while the component is in `Testing Phase`, any of the components it depends on suddenly goes into `Implementation Phase`, then the testing should be stopped in order not to waste the testing effort on outdated components. Lastly, if the component is in `Ready to Finalize` state, and all the components it depends on are `Finalized`, then the component can be finalized.

Advantages. This use case demonstrates the suitability of the modeling approach for expressing of coordination of project teams towards ensuring consistency and correctness of a complex product. It shows expressiveness of our modeling notation comparing to existing modeling approaches that would capture process coordination either in a text form or via events (See Sec. 3).

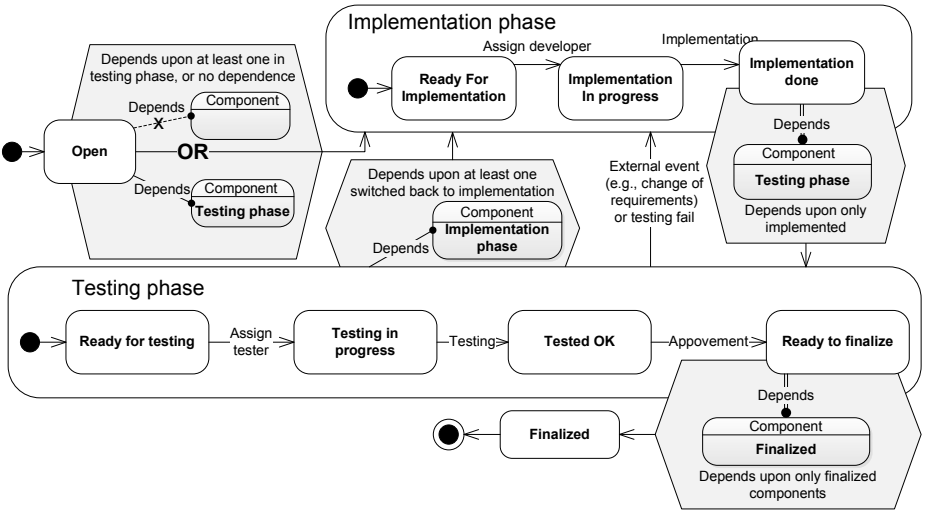

Fig. 6. Use case - dependent components

6 Discussion

Advantages of the modeling approach were demonstrated in the previous section. However, we perceive that our model also has particular disadvantages. Absence of explicit communication entities (events or messages) in the modeling approach is a strength, but also a weakness. A modeler cannot immediately see what parts of a business process (states) other processes rely upon. Given that definitions of events and messages represent a process interface, a modeler will not be able to remove or change process states without a risk of affecting other models. However, this problem can be remedied with State clustering available in statecharts.

We envision, that the discussed visual query language might need additional elements for greater expresiveness. For instance, aggregation elements to express query of exact number of neighbors residing in particular state, or aggregated state of all neighbors. However, in this paper we focused on fundamental concepts, thus trying to keep the appropriate level of detail.

7 Conclusion

This paper proposes a modeling approach and a corresponding graphical notation for creative human collaboration processes. The applicability of the approach was demonstrated through several use-cases, and its strengths and weaknesses were discussed.

Comparing to existing approaches, our contribution has two main distinguishable features: it is capable of capturing specific conditions in form of patterns in

related artifacts of the process, and it advocates a communication model where a process can modify only its own state and cannot explicitly impact the related processes. We have shown that these features are naturally suitable for modeling of collaboration processes. Although our approach was designed with this focus, we do not exclude its applicability in other areas.

Our future work includes the development of an associated execution framework and the integration with existing business process technologies and collaborative software.

References

1. van der Aalst, W.M., Barthelmess, P., Ellis, C., Wainer, J.: Workflow modeling using proclets. In: Scheuermann, P., Etzion, O. (eds.) CoopIS 2000. LNCS, vol. 1901, pp. 198–209. Springer, Heidelberg (2000)
2. van der Aalst, W.M., Weske, M., Grnbauer, D.: Case handling: a new paradigm for business process support. Data & Knowledge Engineering 53(2), 129–162 (2005)
3. Barthelmess, P., Wainer, J.: Workflow systems: a few definitions and a few suggestions. In: Proceedings of Conference on Organizational Computing Systems, COCS 1995, pp. 138–147. ACM, New York (1995)
4. Bhattacharya, K., Hull, R., Su, J.: A data-centric design methodology for business processes. In: Handbook of Research on Business Process Modeling, ch. 23, pp. 503–531 (2009)
5. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Conceptual Modeling of Workflows. In: Papazoglou, M.P. (ed.) ER 1995 and OOER 1995. LNCS, vol. 1021, pp. 341–354. Springer, Heidelberg (1995)
6. Consens, M.P., Mendelzon, A.O.: Graphlog: a visual formalism for real life recursion. In: Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1990, pp. 404–416. ACM, New York (1990)
7. David Harel: Statecharts: a visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (1987)
8. Dickson, G.W., DeSanctis, G.: Information Technology and the Future Enterprise: New Models for Managers. Prentice Hall PTR, Upper Saddle River, NJ, USA (2000)
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. Engels, G., Groenewegen, L.: Towards Team-Automata-Driven Object-Oriented Collaborative Work. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing. LNCS, vol. 2300, pp. 247–255. Springer, Heidelberg (2002)
11. Hagen, C., Alonso, G.: Beyond the black box: event-based inter-process communication in process support systems. In: Proceedings of 19th IEEE International Conference on Distributed Computing Systems, pp. 450–457 (1999)
12. Latella, D., Majzik, I., Massink, M.: Towards a formal operational semantics of uml statechart diagrams. In: Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS), p. 465. Kluwer, B.V. (1999)

13. Martinez-Moyano, I.: Exploring the dynamics of collaboration in interorganizational settings. In: Creating a Culture of Collaboration: The International Association of Facilitators Handbook 4, p. 69 (2006)
14. Müller, D., Reichert, M., Herbst, J.: Data-Driven Modeling and Coordination of Large Process Structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)
15. Neumann, J.V.: Theory of Self-Reproducing Automata. University of Illinois Press, Champaign (1966)
16. Nurcan, S.: A survey on the flexibility requirements related to business processes and modeling artifacts. In: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences, HICSS 2008, pp. 378–388. IEEE Computer Society, Washington, DC (2008)
17. Powell, A., Piccoli, G., Ives, B.: Virtual teams: a review of current literature and directions for future research. SIGMIS Database 35, 6–36 (2004)
18. Reiter, R.: On closed world data bases, pp. 300–310. Morgan Kaufmann Publishers Inc., San Francisco (1987)
19. Sanz, J.: Entity-centric operations modeling for business process management - a multidisciplinary review of the state-of-the-art. In: 2011 IEEE 6th International Symposium on Service Oriented System Engineering (SOSE), pp. 152–163 (December 2011)
20. Wieland, M., Kopp, O., Nicklas, D., Leymann, F.: Towards context-aware workflows. In: CAISE 2007 Proceedings of the Workshops and Doctoral Consortium. Citeseer (2007)