An Ontology for ActionCenter-oriented Collaboration Platforms

Azamat Mametjanov, Cuong Nguyen, Douglas Kjeldgaard, Robert Briggs
Center for Collaboration Science
University of Nebraska at Omaha
{amametjanov, cdnguyen, dkjeldgaard, rbriggs}@unomaha.edu

Abstract
There is a growing number of collaboration technology platforms that support rapid development and deployment of ActionCenter applications – collaborative applications that encapsulate both collaboration expertise and tools for effective collaborative work practices using the facilitator-in-a-box strategy. This strategy enables instantiation and diffusion of state-of-the-art collaboration patterns for high-value recurring tasks. A side effect of the growing number of platforms is the potential for incompatibilities among ActionCenters that can reduce interoperability, knowledge sharing and reuse. We present an ontology for ActionCenter-oriented collaboration platforms that formalizes key concepts of the approach using OWL. The resulting ontology can reduce ambiguities and promote knowledge sharing, reuse and standardization.

1. Introduction
Collaboration – a joint effort toward a common goal [1] – has become increasingly important to the success of many organizations [2]. However, it can be a mixed blessing due to a complex set of communication, coordination, and co-production challenges [3]. Studies show that usage of Group Support Systems (GSS) and other collaboration technologies (collectively groupware) can substantially improve group efficiency and effectiveness [4-8]. Teams using groupware can attain 50-70% reductions in labor hours and can cut project cycle times by 70-90% [9-11].

While groupware technology offers new opportunities to teams, it has been slow to transition into the workplace due to the lack of group process expertise [12, 13]. Groups supported by professional facilitators – group-process experts who design and conduct collaborative work practices – can be significantly more productive than groups without them [14-17]. Unfortunately, professional facilitators are scarce and too expensive to retain within an organization’s personnel [18, 19]. Moreover, when internally trained facilitators leave their positions, the knowledge they have accumulated disappears with them, leaving nobody behind who can champion and use the technology in ways that create value [10, 18].

One of the strategies for accumulating organizational knowledge is to codify the work practices used by the experts for training of new or inexperienced personnel. An example of this strategy is the facilitator-in-a-box approach, where fully documented, stand-alone collaborative applications or ActionCenters encapsulate effective work practices and guide teams through step-by-step group activities [20]. This enables efficient state-of-the-art collaboration in recurring group tasks.

The growth of this approach can be a mixed blessing because it can lead to ambiguity and interoperability problems when deploying on one platform an ActionCenter developed on another platform. Therefore, the goals of this research are to
1. Provide a shared knowledge model of ActionCenter applications for use by collaboration engineering community;
2. Improve the communication within the community by developing shared language and terms;
3. Define and analyze collaboration engineering concepts, attributes and relationships;
4. Improve interoperability among collaboration platforms by standardizing ActionCenter formats.

2. Background
Ontology is an explicit representation of knowledge as a set of concepts within a domain and the relationships between the concepts [25]. The primary uses of ontology are to describe the domain and to reason about the entities within the domain.
Exemplary applications of ontology include such areas as agent-based artificial intelligence, semantic web, biomedical informatics and enterprise information exchange among others.

The primary components of ontology are concepts, attributes and relationships. Individuals, which are instances of concepts, can be included into ontology to define and enumerate the most basic instances. Rules and restrictions can also be included into ontology to define assertions about ontology components. Thus, concepts, attributes, relationships and assertions can collectively describe both the syntax and semantics of a domain.

The domain of ActionCenter-oriented collaboration engineering uses a two-staged architecture. In the first stage, a Collaboration Engineer (CE) uses a Computer-Assisted Collaboration Engineering (CACE) tool to design an effective work practice by defining the content and sequence of collaborative activities. The activities are packaged into a collaborative application (ActionCenter), which is made available for the public to use. In the second stage, practitioners facing a group task or challenge obtain an ActionCenter that defines an effective work practice of dealing with the challenge. They deploy the application on a Collaboration Support System (CSS) and collaborate within the ActionCenter by following pre-defined patterns of collaboration in an autonomous mode without any aid from professional facilitators.

3. Ontology Development Methodology

For the purposes of developing the ontology, we have used the SABiO (Systematic Approach for Building Ontologies) methodology [26]. This methodology proposes a process with the following life cycle stages:
1. Define goals of ontology;
2. Capture the components of ontology;
3. Formalize the ontology;
4. Reuse and integrate with existing ontologies;
5. Evaluate the ontology;
6. Document the ontology.
Each of the stages has a feedback loop back to the previous stage resulting an iterative improvement process.

We have started the development of the ontology with the goals of clearly defining and standardizing the facilitator-in-a-box approach. For the purposes of capturing the ontology, we have used a collaboration expert with a substantial experience of designing and deploying collaborative applications. Our process of eliciting the components of ontology consisted of weekly meetings for four months during which we used open-ended questions for the expert to describe various components of ActionCenters. The responses were immediately entered into ontology editor Protégé [27] during the meetings. Following the meetings, the entered responses were further formalized and edited for correctness and validation purposes. Due to lack of existing ontologies related to the facilitator-in-a-box strategy, we did not reuse or integrate existing domain ontologies. For the purposes of evaluation, we have implemented a prototype functionality that can export and import an ActionCenter with the elements of the captured ontology. We defer the discussion of the details of the implementation until after the presentation of the ontology. Finally, the ontology was documented and placed online for public use at [29] in the form of a .owl source file that consists of 4200 lines of code [30].

4. ActionCenters Ontology

To capture and formalize the ontology we have used the well-known ontology editor and knowledge acquisition system Protégé [27]. This system provides various ontology formats including XML, RDF (Rich Description Framework) and OWL (Web Ontology Language). For greater interoperability, we have chosen the W3C-endorsed standard representation for ontologies for the web – OWL. We discuss the most salient aspects of the ontology next. Additional details can be found in [30].

Overview Figure 1 provides a visualization of the top-level concepts or entities in the ontology. This view has been generated by the OntoGraf plug-in. It displays the direct specializations (subclasses) of the most general entity in the ontology – Thing. For example, an ActivityModule is a Thing. Similarly, a ShareableObject is a Thing. However, an ActivityModule is not a ShareableObject or vice versa.

Activity modules The core concept of an ActionCenter is its set of phases and activities, which are collected under the ActivityModule entity. Figure 2 summarizes this entity’s relationships: is-a relationships in purple and has-a relationships in yellow. In particular, an activity module has various properties such as a name and an id; leader and follower roles; and agenda items, which can be an activity or a phase that has other phases and activities within it.

There are three specializations of an activity module: ThinkLet, AgendaPaletteItem and ActionCenter. A thinkLet is a packaged facilitation
technique that collaboration engineers can instantiate during collaborative process designs to obtain an effective pattern of collaboration [21-24]. On the other hand, an engineer can create and save new activity modules on an agenda palette as part of a creative experiment. The final product of combining new and existing facilitation techniques is an ActionCenter to support a complete collaborative work practice.

Figure 1. Top-level ontology entities in a subsumption visualization graph: e.g. ActivityModule is a Thing, where Thing is a generalization of all entities.

Figure 2. Activity module and is-a (purple) and has-a (yellow) relationships. An ActionCenter is an activity module in three manifestations. Each activity module has phases and activities along with leader and follower roles arranged according to best practices/thinkLets.
There are three manifestations of an ActionCenter. In the design stage, an ActionCenter exists as a project in a collaboration engineer’s studio. Here, an engineer is free to modify a project and share it with other engineers to collectively design a work practice. Having finished the design, the engineer can publish the project to a public collaborative application store (AppStore), where the project manifests itself as an application. Then, practitioners facing the need addressed by the application can acquire and instantiate it in a process support system (PSS) as a workspace, where the actual collaborative tasks can take place.

Collaboration Support System The underlying technology platform implementing the facilitator-in-a-box strategy is called a Collaboration Support System (CSS). Its relationships are summarized in Figure 3. The two primary functions of a CSS are to support the design and deployment of ActionCenters. In the deployment stage, a Process Support System (PSS) is used by collaboration practitioners to acquire applications from a named app store and instantiate them as workspaces. Note that every PSS must have at least one app store, which is indicated in this figure by brown arrows, whereas optional relationships are indicated by yellow arrows.

In the design stage, a Computer-Assisted Collaboration Engineering (CACE) studio is used to develop new applications as projects. For this purpose, a Collaboration Engineer (CE) has access to seven kinds of palettes and six kinds of editors, which can be used in logical and physical designs of a collaborative work practice. Palettes are displayed in various editor contexts and serve as buffers to save-to and instantiate-from various design artifacts.

Logical Design The design of an ActionCenter starts from creating a new project and launching an instance of ActionCenterBuilder (ACB) to edit the logical design of a work practice. Here, a CE builds a sequence of phases and activities, a set of group roles and a collection of screens, which will be displayed to users in a given role in a given activity. The primary goal in the logical design is to arrange activities and their corresponding screens in a way that focuses users in a particular role to the collaborative tasks at hand.

Figure 4 summarizes the ontological relationships of ACB. First, an ACB is displayed with three kinds of palettes to enable experimentation with various agendas, thinkLets and screen designs. Next, an ACB edits agenda items, which can be activities or phases that contain other phases or activities. Another editable entity is an ActionCenter role, which can be a leader or a follower role. The final editable entity is a screen, which appears at the intersection of an activity and a role – an activity-role combination. Note that since a screen can be displayed for multiple activity-role combinations, there exists a screen indirector, which points to an actual screen object. Such indirection reduces data duplication by allowing multiple pointers to a screen object and improves data propagation where a change to the screen object is visible to all pointers of the screen.
Figure 4. ActionCenterBuilder (ACB) edits the logical design of a work practice by sequencing activities and screens in proper activity-role combinations.

Figure 5. Screen editor defines the physical layout of a screen. A screen can have a screen object, which is either a splitter or a tool/control. A vertical/horizontal splitter can have other screen objects within it.

Physical Design Upon completion of the logical design of a work practice, a CE can turn to the physical design of individual screens to be displayed to group members during a collaborative session.

Figure 5 displays screen editor entities. An engineer can split a screen horizontally or vertically and place tools, controls or other splitters within the new screen regions.
Figure 6. Tools editor entities. A tool consists of tool elements -- either a component or a shell. A parent-child shell has at most one parent component and zero or more child tool elements. An outliner component has header, input and output panels. User-submitted contributions appear in the output panel according to engineer-specified population rules. A component can have other tool elements as subcomponents. A population rule can have other population rules as subordinates.

A screen editor (not pictured) displays with screen, control and tool palettes. This allows an engineer to instantiate existing objects within a screen region or to save newly configured objects back to the palettes.

**Tool design** Upon completion of the screen layout, an engineer can launch a tool editor to design a tool within a given screen region. Figure 6 summarizes the ontology of tool-related entities.

A tool consists of tool elements with at most one root-level element. By adding other elements as subordinates to the root-level element, an engineer provides users with a capability to launch new tools from within the root-level tool. At the moment, there are two kinds of tool elements: components and shells. A parent-child shell has at most one component as a parent element and zero or more elements as children. Other kinds of shells such as a tab shell (not shown) may have different configurations.

A tool component can have header, input and output panels. Contributions submitted by users within an input panel are displayed in the component’s output panel. For example, in an outliner component users submit text-based contributions that are displayed in an outliner view.

The content of a component’s output panel is populated according to the engineer-specified population rules. The rules define the type of user-submitted contributions and their relationship to the component such as a child-of or member-of relationships. A rule can have another rule as a subordinate to create hierarchical user contributions.

**Shareable objects** During the design or deployment of applications, a user may need to share an object with others. Figure 7 summarizes the shareable objects. Each palette within a CACE studio has folder’s called categories that can be shared with others. For example, an engineer can place public icons in a folder and invite other engineers to view and edit the icons. Similarly, a facilitator/practitioner may give access to other facilitators to the apps within a given application library.

Each shareable object has a sharing editor where the owner of an object can invite other users with read and write privileges. An ActionCenter project also has a sharing editor for an engineer to give access to other engineers.

**Other editors** The remaining kinds of editors are element editor and report editor. An element editor provides various input form elements such as text, pick-list and icon upload fields to edit various properties of ActionCenter entities. A report editor provides text fields, where an engineer can provide descriptions of ActionCenter entities. These descriptions can be extracted from a project to create reports about the project.
5. Discussion

The ontology described in the previous section has been captured in Protégé and formalized in description logic-based language OWL. One of the built-in capabilities of Protégé is to launch a semantic reasoner on an ontology to validate the consistency of assertions made about the components within the ontology. We have used HermiT semantic reasoner to check the ontology for inconsistencies. No errors or inconsistencies have been reported by the reasoner.

To put the ontology into practice we have also implemented an export-import functionality on an existing ActionCenters platform [28] that allows an engineer to save and reload applications to/from XML files. The format of XML files is in direct correspondence with the structure and terminology of the ontology. Thus, not only the ontology serves the purposes of knowledge sharing, reuse and standardization, but also it can be used to achieve interoperability among different ActionCenter-oriented collaboration platforms. For example, an application can be built on one collaboration platform/server, exported and shared with colleagues, who can import the application on a different platform/server and reuse existing application content.

6. Related work

**Ontology and collaboration** In the literature, the word “ontology” is often found along with the word “collaboration”. In fact, an important function of an ontology is to create some common ground based on which people or machines collaborate with one another [31]. For example, in the biology area, a gene ontology has been created to serve as a common vocabulary for genes and their products, which was important to the interoperability among various genomic databases [32]. In the ubiquitous computing field, Christopoulou and Kameas [33] proposed GAS ontology that described the basic concepts in the ubiquitous computing environment and their inter-relations to enable the communication and collaboration among different devices in the environment. Ontologies were also applied to support the collaboration among heterogeneous government organizations in public policy making, implementation and evaluation [34].

ActionCenter-oriented ontology is created with the same purpose as the example ontologies above i.e. creating a common language to support collaboration among people and interoperability among different platforms. However, the content of the ontology itself is also about collaboration domain. In other words, the ActionCenter-oriented ontology’s
purpose is to support collaboration and communication in collaboration domain.

Moreover, we also should distinguish between collaboration ontology and collaborative ontology. While the former refers to an ontology describing collaboration domain, the latter refers to an approach in ontology engineering that involves multiple people in the development of ontology [35, 36]. ActionCenter-oriented ontology is about collaboration ontology, rather than about a collaborative ontology engineering method.

**Ontologies of collaboration domain** Because creating and using technologies to support effective and efficient collaboration among people has been an important research goal for many years, efforts have been made in creating ontologies that captured and standardized the expertise and knowledge of the domain. Below, we compare and contrast our ontology with a number of similar works that have been proposed in the literature with the goals of further clarifying the goals and scope of our ontology.

To facilitate the integration of different collaborative technologies into organizations, Oliveira et al. [37] proposed a collaboration ontology to characterize the common elements of collaboration processes. The ontology was the further elaboration on the 3C Model suggested by [38], which argued that a collaboration process was composed of three main components: the *Cooperation* – “joint effort in a shared space to achieve some goals”, the *Communication* among individuals to support their cooperation, and the *Coordination* that defined the rules and protocols based on which the cooperation and communication among participants would be enacted. The cornerstone of the cooperation component was the collaboration session that had objectives, followed certain protocols, could consume or generate collaboration artifacts, happened in sites, and involved at least two participations performed by participants. The communication component included communication actions that involved send and receive events of messages expressed in certain language among participants. The coordination component had not been elaborated.

Compared with Oliveira et al.'s work, our ontology also contains elements of collaboration processes suggested in their ontology. Specifically, the concept of collaboration session is equivalent to the concept of workspace in our ontology. The coordination rules and protocols are embedded in the roles that participants have in a workspace and the population rules for the contributions. However, our ontology does not only stop at giving the descriptive elements of a collaboration process. Rather, as an ontology about a framework for creating and executing collaboration processes, our ontology also views a collaboration process as a concrete product that appears through a production life cycle in at least three forms: as a project that is created by a collaboration engineer, as an application that users can buy, sell or share, and as a workspace that is a specific instantiation of an application.

To support classification of various collaboration technologies, Ellis and Wainer [39] proposed a conceptual model of groupware. According to this model, a groupware system could be characterized by three complementary sub-models: the ontological model that described the objects existing in the system and the operations that users could perform on the objects (e.g. view, create, modify, destroy), the coordination model describing the activities and the coordination of the activities that users would perform to complete their collaborative tasks, and the user-interface model describing how the system supported users to view information objects, view other participants and view the contexts in which the collaboration happened.

In relation to Ellis and Wainer’s conceptual model, our ontology represents a framework that can be classified as meta-groupware, since the framework’s purpose is to generate groupware applications. However, the type of groupware applications that the framework generates contains richer information than the ontology in [39], which mainly focuses on the capabilities that a groupware system can provide to users. An application created in the ActionCenters framework, besides the functionalities it provides to users, also embeds the specific collaborative goals supported by the rationale represented by a sequence of best practices/thinkLets.

Pattberg and Fluegge [40] realized that while collecting and using collaboration patterns – “a kind of documentation for a proven collaboration solution” was advantageous and collaboration patterns had been understood in different ways and documented at different levels of abstraction. To support consistent collection of collaboration patterns, the authors created an ontology of collaboration patterns [40]. According to this ontology, collaboration patterns could be classified into one of the five layers of abstractions: *abstract collaboration patterns* addressing collaboration goals (e.g. group building, decision making); *collaboration patterns* layer providing proven solutions for collaboration problems (e.g. argumentation, negotiation); *lower collaboration patterns* composing of reusable collaboration building blocks (e.g. edit information, publish information); *collaboration
services providing reusable implementation components such as shared white-board, user list; and at the lowest level of abstraction communication technologies layer listing the base technologies such as Ajax or Web services.

The idea of collaboration patterns is also incorporated in the ActionCenter ontology, which is manifested by the ThinkLet component. However, the purposes of our ontology and Pattberg and Fluegge’s ontology are fundamentally different. While their ontology aims at providing a taxonomy for classification of collaboration patterns, our ontology is a description of a process that turns collaboration patterns into concrete groupware applications.

7. Conclusion

This paper advances the facilitator-in-a-box strategy of diffusing state-of-the-art collaboration patterns into practice by defining a domain ontology for collaboration platforms. The ontology defines the key concepts, attributes and relationships in a formal description logic-based language. The primary benefits of the ontology are the promotion of knowledge model sharing in the collaboration engineering community, standardization of terminology and improvement of interoperability among different collaboration platforms.

Future work in the engineering of the domain ontology includes extensions that cover collaboration capabilities beyond those presented here. For example, new collaboration tools and components need to be described and captured in the ontology. Another aspect of future work is to enrich the ontology by including additional cardinality constraints on relationships among entities or by adding new kinds of entity relationships.

8. References


