A Uniform Meta-Model for Mediating Formal Electronic Conferences

Jia Zhang
Department of Computer Science
Northern Illinois University
DeKalb, IL 60115
jiazhang@cs.niu.edu

Carl K. Chang
Department of Computer Science
Iowa State University
Ames, IA 50011
chang@cs.iastate.edu

Jeff Voas
Cigital
21351 Ridgetop Circle
Dulles, VA 20166
voas@cigital.com

Abstract

Formal Electronic Conferences (FEC) refers to online meetings for a geographically distributed group of people that are regulated by a rigorous set of rules. FEC technologies enable organizations to replace face-to-face business meetings with trustworthy virtual online meetings. In this paper we present a Robert’s Rules of Order (RRO)-compatible, motion-driven discussion-thread-centered meta-model, which is capable of uniformly modeling formal electronic conference activities. A tailored computerized mechanism, the Collaboration Description Language (CODL) and its runtime environment, is also developed to formalize the model. The CODL virtual machine adds a layer of encapsulation that decouples FEC applications from underlying platforms; therefore, the development of FEC applications will become more reliable, efficient, and secure. Our preliminary experience with this meta-model is also reported.

Keywords:
Formal electronic conferences, Robert’s Rules of Order, uniformity, meta-model

1. Introduction

We live in a society that is run by meetings [3]. According to the National Statistics Council, 37% of employee time is spent in meetings [17]. Since some members of organizations are remotely located, it has been necessary for people to travel, incurring high meeting costs. The significance of meeting-related costs is revealed by MCI [12], a global leader in business communications, in their annual survey Meetings in America. As an example, for a five-person meeting with four attendees traveling by air, the combined hard and soft costs were $5,197.50. As a result, American organizations have been gradually adopting virtual collaboration technology. MCI’s 2003 annual report released on October 6, 2003 discovered that 74% of business travelers had used travel alternatives, and 87% expected to use audio, video or Web conferencing in 2004 [11].

Consequently, recent years have witnessed the introduction of a number of new Internet-based conferencing systems, some of which have been used extensively, such as GMD FIT’s Basic Support for Cooperative Work (BSCW) [2], Microsoft’s NetMeeting system [14], MCI’s web conferencing systems [10], Apple’s QuickTime Conferencing [1], WebEx’s WebEx Meeting Center [18], etc. Even with the variety of Internet conferencing systems available, however, a significant number of corporate planners still have concerns over this emerging means of virtual collaboration. MCI’s 2003 survey found that 87% of the respondents still prefer on-site meetings [11]. We will now explore three issues that hinder corporate planners from believing that they can resort to electronic means to facilitate business.

First, other than providing a shared environment, current Internet conferencing systems do not guarantee the order of cooperation nor do they enforce an agenda or guarantee the productivity of the collaboration. Traditionally however, much of business collaboration is governed by “formal” communication rules, and these rules cannot be guaranteed by ad hoc tools. These collaborations require virtual communities that provide moderators and support a rigorous set of rules [15].

Second, existing collaboration-technology products do not address certain key business enablers that can convince corporate planners to believe that they can resort to electronic means to facilitate business. Security and reliability must be considered as of paramount importance in the electronic medium.

Third, existing systems often sustain low interoperability. Corporations must learn differing collaboration metaphors and styles in order to work with different groups, and even when the groups are co-located.

Our research is further motivated by the fact that existing collaborative systems suffer from low reusability, flexibility, extensibility, and adaptability [9]. They generally lack a sound infrastructure to support the development of a wide variety of collaborative applications governed by rules. As a result, developers often resort to ad hoc strategies for establishing various collaborative systems. Meanwhile, it is hard to verify that the constructed systems are secure and function as desired.

The development of efficient and adaptive electronic collaboration governed by nonambiguous rules requires methodologies that are beyond the current state of the art.
Therefore, in this paper we propose a uniform meta-model approach to support the creation of complex and diverse formal electronic meeting systems. In our definition, Formal Electronic Conferences (FEC) refers to online meetings performed by a geographically distributed group of people with common interests. An FEC system is enforced by a set of formal collaboration rules, a set of well-defined sequential steps that must be followed by each collaborator during his/her cooperative work until a goal is reached. An FEC ensures secure and traceable collaboration so that the cooperative work can be replayed and recovered while allowing for confidential collaborative information to be hidden. An FEC system also authenticates the outcomes of the collaboration.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. In Section 3, we present our meta-model. In Section 4, we discuss the implementation of the meta-model. In Section 5, we introduce CODL. In Section 6, we discuss CODL runtime environment. In Section 7, we discuss experiments and evaluations. In Section 8, we present conclusions and discuss future work.

2. Related work

The last decades have witnessed a variety of research effort on coordination models and languages supporting parallel and distributed computing [13]. Many role-assigned actor-based models are built to model collaboration work. COCA [7] separates coordination policies from computation. Multitel [4] describes a service architecture that defines the user interaction logic for specific services. ADOME [8] defines roles as the social commitments or obligations of members in a group. Hawryszkiewycz presents a metamodel structure, in which the central component is the workspace where activities take place [5]. These models aim to facilitate the communication and coordination among multiple concurrent computing entities [19]. In contrast to their work, our approach focuses on modeling the basic construct of process control in formal meetings.

Yang proposes a uniform meta-model to uniformly model different cooperation scenarios [19]. The basic construct of his meta-model is an activity that describes a piece of work to be finished by certain entities in a certain way under certain constraints. Object Description Language and Cooperation Description Language are tailored to formalize the model. In contrast to his work aiming at generic collaboration, our approach targets collaboration in FEC, which require formal rules enforcement. Therefore, our model and tailored language focus on efficient, secure, and reliable meeting collaboration modeling.

A significant amount of research work has focused on a formal treatment of collaboration rules. Among the variety of reported work, COCA [6] formalizes a small subset of RRO [7] with a simple single floor control and a simple voting strategy; Smith and colleagues [16] present an organized Internet chat environment, in which a simple protocol for floor control is adopted and the conversation history is recorded in a threaded log; ADOME [8] integrates versatile coordination policies into a collaboration by dynamically binding social commitments to different roles. However, these prototypes of collaboration rely on simplified floor-control models. In contrast, we will base our approach on Robert’s Rules of Order in order to lay down a formal rule-based collaboration control mechanism, but we will allow for limitations and adaptation to the particular needs of various electronic settings. To the best of our knowledge, researchers in CSCW have not paid adequate attention to such issues.

In addition, most existing collaboration systems impose a set of rules for users to “follow,” such as [1,18]. We have paid particular attention to accommodating the dynamic and evolutionary nature of human interactions, where rules can be dynamically changed on the fly and interpreted. Some related work does allow dynamic rule generation, such as that in [19]. However, his rule generation is fully dependent on users. In contrast, our dynamic rule generation is compatible to RRO; therefore, it is more reliable to meeting control.

3. The meta-model

The essential idea of our meta-model is the concept of a discussion thread that carries all the meeting activities and organizes them in an orderly manner. Intrigued by the essential features of traditional RRO, we define a discussion thread representing a single semantically cogent path through the meeting space. Just as motions trigger all meeting activities in RRO, our discussion thread model is motion-driven. In this paper, we use event and motion interchangeably. Our meta-model is stated to be a uniform mechanism for FEC in the sense that it is capable of modeling different collaboration scenarios, i.e., synchronous, asynchronous, parallel, serial, and nested collaboration, in a RRO-compatible uniform way with flexibility. In accordance with our meta-model, a formal meeting can be modeled mathematically as a set of interrelated discussion threads, each carrying a piece of work that contributes to the goal of the meeting.
A discussion thread in our model is defined as an active execution container for meeting-related activities. Figure 1 illustrates the structure of a discussion thread. As shown in Figure 1, a discussion thread is an execution container in the sense that it holds the information related to a particular piece of collaborative work (via the control block), and enforces the rules that control the access to the information (via the workflow engine). It is active in the sense that (1) a discussion thread contains a running thread; (2) it allows participants to conduct discussions when it is active; (3) under certain circumstances, it will generate output messages actively to its external environment, which other discussion threads may rely on; and (4) each discussion thread maybe in one of the following six states: new, ready, running, pending, completed, and aborted.

Just as each discussion thread represents a piece of work, a meeting as a whole represents a larger piece of work. The state of a meeting is naturally defined as the combination of all the states of its component discussion threads. It records how the agenda items are processed and what the latest results are. In this sense, a meeting can also be regarded as a discussion thread, which can then be used as a component of another meeting. Therefore, meeting definitions can be nested to form a hierarchical structure. This feature also proves the uniformity of our meta-model.

### 3.1 States of discussion threads

A discussion thread is triggered by a motion or amendment and ended when a decision is made. When a discussion thread executes within its life cycle, it changes state. The state of a discussion thread is defined in part by the current activity of that discussion thread. Each discussion thread maybe in one of the following six states: new, ready, running, pending, completed, and aborted.

The state diagram corresponding to these states is presented in Figure 2. When a motion on an agenda item or an amendment is put on the floor, and corresponding resources (e.g. participants) are identified, a discussion thread is created, or its state is new. When the motion gets seconded, the discussion thread enters the ready state, which means that participants can start to discuss in the context of the discussion thread. When participants start to conduct discussions, the state of the discussion thread is defined as running. We differentiate between ready and running states to increase flexibility, where different strategies can be applied to decide whether the ready state can enter the running state implicitly or requires explicit triggering. If a discussion thread needs to wait for some events to occur, the state of the discussion thread is considered to be pending. When the waiting events happen, the discussion thread returns to the ready state, and participants can resume their discussions. When a discussion thread has finished its execution with a decision, it is considered to be completed. If a discussion thread fails, then the modifications to the related resources need to be rolled back, and the discussion thread is considered to be aborted.

As shown in Figure 1, each discussion thread has an associated control block for data storage. Only the data maintained by a running discussion thread are accessible. It should be noted that the transitions between states are driven by events; however, for simplicity, the events that trigger the transitions between states are not shown in Figure 2.

### 3.2 Dependencies between discussion threads
The meeting space may have concurrent discussion threads running simultaneously. It is inevitable that two or more discussion threads may have some time constraints between them, e.g., one discussion thread may be terminated only after another one is terminated. We use the term dependency to indicate the relationship between discussion threads. Instead of establishing direct and tightly coupled links between inter-dependent discussion threads, their dependencies are established through a relatively loose publisher/subscriber model. A discussion thread that is dependent upon another discussion thread subscribes to the latter to receive event notifications when changes occur to the latter. This strategy conforms to the object-oriented design paradigm and handles dynamic dependencies more robustly.

There is another kind of dependency relationship between discussion threads. A discussion thread may have a participant propose an amendment, in which case it creates a new discussion thread. The relationship between these two discussion threads is a parent-child relationship, which is also important to maintain to regulate proper life cycle state changes.

### 3.3 Control blocks of discussion threads

A discussion thread not only has the features of the execution states, but also carries a set of quantitative properties. For example, a specific discussion thread identifies itself at a specific time point by its objective, its participants, its state, etc. With these data storages, the actions performed by the participants can be tracked; therefore, latecomers can retrieve the information before they join, and all the actions can be rolled back if it is decided to abort a discussion thread. Therefore, a discussion thread is represented by a discussion thread control block (DTCB). A DTCB contains different pieces of information associated with a specific discussion thread at a specific time point, which includes the following:

- **Discussion thread state**: The state may be new, ready, running, pending, completed, or aborted.
- **Participants**: This information identifies the group of people who have access to the discussion thread, and their roles may define different access policies.
- **Collaboration rules**: This information specifies how the discussion thread will execute in its life cycle, and will be discussed in more detail in the next section.
- **Delta-documents**: This information stores the historical versions of the shared documents.
- **Child discussion threads**: This list records all the direct discussion threads spawned by this discussion thread.
- **Registered discussion threads**: This information indicates the set of other discussion threads that depends upon it.
- **Subscribed discussion threads**: This information indicates the set of other discussion threads that this one depends upon.

In summary, the DTCB simply serves as the repository for any information that may vary from discussion thread to discussion thread.

### 3.4 Rules of discussion threads

In order to effectively and efficiently govern collaboration, we began from a core set of basic rules common to a wide range of FEC applications and provided mechanisms to allow run-time modifications of rules for adaptation to dynamic scenarios. Instead of attempting to invent generic collaboration rules, we established a library of collaboration primitives based on well-established RRO [15], since RRO is the predominant mainstream approach to meetings in US [17].

We examined RRO to identify superfluous rules to discard, and explored necessary extensions so as to exploit the newer capabilities of electronic media. The examples of rule primitives are the following: propose a motion, propose a vote, second a motion, etc. Due to the page limit, here we will not discuss in detail our rule primitive library. With the library of collaboration rule primitives, new collaboration rules can be dynamically constructed to adapt to the ever-changing business requirements. It is in this sense that we call the discussion thread a uniform model. Certainly a set of meta-rules is required to ensure that the newly constructed rules conform to the basic RRO logic.

In order to effectively manage the state changes of discussion threads, certain types of additional rules need to be identified at the creation time of the discussion threads and enforced throughout their entire life cycles. Here we classify three categories of compulsory rules. The first type of rules specifies when and under which circumstances a discussion thread will change its state, or how the decision can be made, i.e., the voting strategy. The second type of rules specifies how shared data can be accessed. The third type of rules specifies how a discussion thread interacts with its environment, for example, how a discussion thread notifies its completion to another discussion thread whose completion depends on it.

### 4. Implementation

Figure 3 shows a typical scenario of Java pseudo code to implement the discussion thread. As shown in Figure 3,
To help formalize our architectural model, we developed a formal description language called “Collaboration Description Language” (CODL). CODL is a high-level description language for engineers who may not be actual developers to precisely specify the requirements of an FEC application. Then, a language translator is developed to automatically generate an architectural skeleton and program code. In the previous section, we mentioned that a library of collaboration primitives is constructed out of the RRO database of rules. Therefore the CODL translator actually translates the CODL specifications into a set of collaboration primitives executed over a Java virtual machine. Another advantage of using CODL to build an FEC application is that this technique enhances interoperability of FEC systems by making it feasible for different FEC systems to interact with each other.

Due to page limitations, the discussions here will be kept as brief as possible. The detailed information will be put in another paper. The syntax of the CODL was defined using the BNF format. Figure 4 illustrates a partial definition of CODL related to the declaration of a discussion thread. The first rule states that a discussion thread definition starts with the keyword DiscussionThread and a qualified name. Optionally, it can derive from a superclass, from which all definitions in the discussion thread body are inherited. The body of a discussion thread definition is composed of two parts: the workflow engine and the discussion thread control block. A discussion control block contains a set of data attributes, each representing one of the following properties: state attribute, collaboration rules, time frame, a discussion thread that is subscribed to, a discussion thread that registers to this one, document versions, participant, historical record of the progress of the discussion thread, and the decision. The state of a discussion thread can be one of the six states: new, ready, running, pending, completed, or aborted. A collaboration rule may be one of the four types of rules: the RRO primitive rule from the primitive rules library, the rule public class DiscussionThread extends Thread{
  ...public run() {
      ...
      if (motion (amend)) {
          suspend(this);
          createDisThread(this);
      } ...
      if (decision(amend)) {
          resumeParentDisThread();
          terminate(this);
      } ...
      ...
      // all attributes
      private DiscussionThread parDiscussionThread;
  }
  ...
}

Figure 3. A piece of discussion thread implementation


5. Collaboration Description Language

DiscussionThreadDeclaration ::= DiscussionThread QualifiedName [EXTENDS QualifiedName]
DiscussionThreadBody;
DiscussionThreadBody ::= { DiscussionThreadControlBlock } { WorkflowEngine };
DiscussionThreadControlBlock ::= DiscussionThreadAttributes;
DiscussionThreadAttributes ::= DiscussionThreadAttribute { DiscussionThreadAttribute};
DiscussionThreadAttribute ::= State | CollaborationRules | TimeFrame | RegisteredDiscussionThreads | SubscribedDiscussionThreads | DeltaDocuments | Participants | Progress | Decision;
State ::= New | Ready | Running | Pending | Completed | Aborted;
CollaborationRules ::= CollaborationRule { CollaborationRule };
CollaborationRule ::= CollaborationPrimitiveRule | StateChangeRule | AccessRule | RegistrationRule;

Figure 4. A Partial Definition of CODL for FEC Systems
about how to change the state, the rule about the access control, and the rule about how to register to and from other discussion threads.

6. CODL runtime environment

A CODL runtime environment is built to dynamically interpret collaboration commands at run time and translate the CODL description into sets of collaboration primitives provided by the underlying collaboration platform. Figure 5 illustrates the CODL runtime environment. As shown in Figure 5, two central components are included: a development environment and a CODL virtual machine. In the development environment, three types of editors are provided serving three types of designers who can contribute to the development independently. Independent of application designers, rule designers can input new bylaw specifications via Rule editor and store in Rules database; and collaboration platform designers can input primitives specifications via Primitive editor and store in Primitives database. Meanwhile, application designers use the CODL editor to design application logic. CODL specifications are inputted via CODL editor and stored in Schema database first for future usages. At runtime, the CODL compiler dynamically fetches CODL specifications from Schema database, and interprets the designs into an intermediate language code that is the composition of a list of collaboration primitives, utilizing the support from the Rules database and the Primitives database. The translated code in the format of the intermediate language is stored in the intermediate database. This set of intermediate language code is then inputted to the CODL generator to generate the real Java code and is stored to the database, when the development phase is finished. The execution environment of our CODL runtime environment contains a Java Virtual Machine (JVM), an event recorder, and a web server. Eventually, the real code will be deployed into JVM and executed, with the event recorder monitoring the execution. Finally the application becomes available to application users via the Web server that acts as a web front-end.

7. Experience and evaluation

In this section we describe a sample of “real-life” projects where our meta-model and CODL were used to model and construct the systems, and elaborate on the lessons learned from such experiences.

CODL was originally conceived to model the Collaboration Net (C-Net) project, a synchronous/asynchronous Web meeting environment that enforces RRO. Our meta-model was used to model both the synchronous and asynchronous modes of the collaboration. Students are divided into two groups, one
group using Java language and waterfall model to develop the system, while the other group using CODL to develop the system. By comparing the time and effort spent by each group, CODL has permitted developers to deliver prototypes very rapidly.

**An on-line auction system** simulates a real-world auction place. When an item is placed for bidding, each user has a chance to bid on the resource; and the person with the highest bid will win the auction. The process of the auction is controlled by some underlying rules and constraints, e.g., each successive bid has to be higher than the current bid on the item. Our meta-model is used to model the auction on items, and CODL is used to develop a prototype of the application. The challenge of the project is that the bidding rules for different items may vary and may change dynamically. The CODL proves to be effective to construct such an application since it has the power to specify different rules and manage changing rules over time.

**A collaborative whiteboard tool (Collwb)** simulates a regular whiteboard where several people can draw sketches. Each user of the collaborative whiteboard has continuous access for viewing the contents of the shared whiteboard, and only a single user can do the drawing at any time. Therefore, a user needs to acquire an exclusive drawing access from the coordinator. The system is responsible for preserving whiteboard consistency across all participating users’ views by synchronizing their contents. The system also monitors the holder of the exclusive drawing rights to assure timely release of the drawing right. Our meta-model is used to model the shared access to the whiteboard and proved to be effective.

One of the original contributions of our experimental projects is the evaluation of several experiences of specifying and developing FEC systems with our meta-model and CODL, side by side with developers trained in different development technologies. In summary, the use of our meta-model and CODL for synchronous/asynchronous FEC systems has proved beneficial in both effectiveness and efficiency.

The CODL virtual machine adds a layer of encapsulation that decouples FEC applications from underlying platforms. Therefore, the development of FEC applications will become more reliable, efficient, and secure. The reliability of FEC applications constructed from the CODL specification stems from the fact that developers will be shielded from complex implementation issues that are involved in building a powerful collaboration system from scratch. Efficiency is guaranteed since the CODL compiler saves the developers of defining and programming each collaboration primitive required by a specific FEC application. Security is improved by the built-in and top-down approach in which the coordination policies are developed with the CODL specification.

The incorporation of security features is driven by our concept of discussion threads. Collaboration activities will be executed as secure collaboration threads running within the CODL Virtual Machine. The CODL sandbox encapsulates collaborative activities securely within the collaboration threads instantiated from CODL Virtual Machine primitives. This method will elevate the security mechanism to the collaboration level and improve overall system security. For example, a conflicting coordination policy between a chair’s privileges and a participant’s rights (e.g., awareness) will be rectified within the CODL sandbox. Additional research is needed concerning this new security measure; however, we believe that it holds promise.

In addition, because our model is based upon RRO, it lays down such a formal rule-based collaboration control mechanism and assures the order of the collaboration. Each discussion thread has a parent thread, which defines the synchronization rules for all its children threads. Furthermore, discussion threads delineated by motions and decisions serve the important purpose of control of flow. This ensures that the flow of the meeting is goal-directed and makes explicit the structure of the flow. Our model is based upon a library of collaboration rule primitives. New collaboration rules can be dynamically constructed adapting to the changing business requirements. Furthermore, the control block of our model keeps track of the cooperation status and history.

In summary, our RRO-compatible motion-driven discussion-thread-centered meta-model and its tailored CODL provide a generic infrastructure that satisfies the requirements that we outlined in the introduction.

However, our experiments have also shown various limitations of the current approach and prompted several changes and extensions. Currently, when a discussion thread spawns a new discussion thread, the whole control block of the parent discussion thread is copied to the child discussion thread. In large-scale systems, this mechanism may result in low performance. In addition, our current model does not allow data values to diverge; therefore, users cannot have the different degrees of coherency for shared data.

**8. Conclusions and future work**

In this paper we present an RRO-compatible, motion-driven discussion-thread-centered meta-model that is capable of uniformly modeling formal electronic conference activities. A tailored computerized mechanism, the CODL language and its runtime environment, is also developed to formalize the model. The uniform nature of our model provides a grounded theoretical framework for reasoning about architectural interactions of formal
collaboration. The CODL virtual machine adds a layer of encapsulation that decouples FEC applications from underlying platforms; therefore, the development of FEC applications will become more reliable, efficient, and secure. Our research results can be applied to support a broader range of collaboration purposes, where discipline and security are not critical to conducting cooperative work.

Our future work includes the following directions. First, we will attempt the formal verification approach to verify the CODL specification of an FEC system. Secondly, we plan to develop tools to mechanically derive test cases to detect deadlock situations and to reason about dynamic behaviors.

9. References
