ARCADE: Action-Centered Rapid Collaborative Application Development and Execution

A. Mametjanov\textsuperscript{1}, Douglas Kjeldgaard\textsuperscript{1}, Todd Pettepier\textsuperscript{1}, Conan Albrecht\textsuperscript{2}, Stephan Lukosch\textsuperscript{3},
Robert Briggs\textsuperscript{1}

\textsuperscript{1}Center for Collaboration Science
University of Nebraska at Omaha
\{amametjanov, dkjeldgaard, tpettepier, rbriggs\}@unomaha.edu

\textsuperscript{2}Information Systems Department
Brigham Young University
conan@warp.byu.edu

\textsuperscript{3}Systems Engineering Department
Delft University of Technology
s.g.lukosch@tudelft.nl

Abstract

Synchronous collaborative applications typically take several programmer-years to develop because of the need for difficult-to-build capabilities required to resolve issues that emerge when multiple users execute potentially conflicting actions on the same data objects. It is therefore too costly to build bespoke collaborative applications for many tasks. Generic applications, however, are often poorly suited to specific requirements of a given task. This paper presents a conceptual implementation of an integrated collaborative system – ARCADE – to design and deploy complete collaborative applications for any problem domain. The system provides orders of magnitude reduction in development time. The resulting applications – ActionCenters – can guide the practitioners through well-designed collaborative work practices without requiring that they become collaboration experts. Adoption of ActionCenters can significantly increase diffusion of state-of-the-art collaboration in organizations.

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.

While there exists a number of collaboration tools that support specific group activities such as audio-video conferencing (e.g. Skype) and shared document editing (e.g. Google Docs or Adobe Connect), few tools support complete collaborative work practices for use in development of collaborative applications. Even if a tool supports a work practice from beginning to end, it typically is hardwired toward a specific goal and cannot be repurposed to other collaborative work practices without major interventions from the tool’s original developers. Therefore, one of the main goals in the research on collaboration technology is to conceptualize general-purpose tools for development and deployment of end-to-end collaborative applications.

In this paper, we present a generic Action-centered Rapid Collaborative Application Development environment built to support synchronous and distributed multi-user engineering and Execution of collaborative applications (ARCADE). Its innovative features are:
• A Universal Data Model (UDM) to dynamically create and store arbitrary relational data in a fixed set of database tables. A Collaboration Engineer (CE) does not have to (re-)design the data model to manipulate new kinds of data.
• Population rules to specify dynamic data types and their relationships during construction of new data models.
• Dynamic contribution channels for event notifications across multiple workstations and multiple user interface components on a single workstation.
• Graph-based manipulation of objects for greater flexibility in handling non-tree-like data.
• Specification of standardized application programming interfaces (API) for interoperability with other groupware systems.

The resulting system enables rapid development of collaborative applications that is $10^4$ times faster than conventional GSS development practices.

2. Architecture

Among the large number of modern collaboration tools, two of the major functional deficiencies are (a) a lack of end-to-end support of collaborative work practices as in tools that support only certain aspects of collaboration such as shared document editing tools; or (b) a lack of generality and the ability to be repurposed to a diverse range of collaborative problems as in tools that focus on specific work practices such as project management or requirement engineering tools. Because of this, facilitators often resort to a collection of tools to design the entire work practice. This requires facilitators to be familiar with a number of tools in order to pick the right one for a work practice at hand. What is really needed is an integrated tool that supports most, if not all, collaboration features such as synchronous document editing while at the same time allowing facilitators to design work practices for different problem domains all within the same system. Below, we outline an approach for handling one of these two problems: lack of end-to-end support for a work practice. The subsequent sections address the lack of generality problem.

The architecture of the facilitator-in-a-box strategy consists of two parts: design and execution [20]. In the first phase, 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 (a Collaboration Support System (CSS) application or ActionCenter), which is made available for the public to use. In the second phase, practitioners facing a group task or challenge obtain an ActionCenter that defines an effective work practice of dealing with the challenge. They collaborate within the ActionCenter by following pre-defined patterns of collaboration in an autonomous mode without any aid from professional facilitators.

2.1. Designing a session

At design-time, the first task of a CE is to define the sequence of actions that lead a group toward their common goal. This flow may be organized into activities and phases that respectively define milestones for a set of actions or a set of activities.

One or more of the activities may follow best practices or may directly instantiate scripted collaboration procedures of thinkLets [21-23]. These practices and procedures may define in detail who should be doing what within a given role at a given stage of a group task. On the other hand, instead of following previously published procedures, the CE may define and experiment with new logical designs and process flows. In either case, the system allows the CE to create new or instantiate existing templates for group activities.

Having defined all logical and functional details of collaboration, the CE turns to the physical design using activity-role, screen and tool editors [20]. The activity-role editor provides a two-dimensional tabular view to specify the screens that should be displayed for a given activity and a given role. The design of a screen is accomplished in the screen editor, which defines the screen’s content and positioning of its elements. Finally, the tool editor allows the CE to compose screen elements from a collection of simple configurable components such as a tree or input form components to create a tool for a certain group task such as brainstorming or voting. All of the editors provide a convenient drag-and-drop capability with WYSIWIG (what-you-see-is-what-you-get) views to give the CE complete control over the user interface including rich-text editing of any textual content.

Upon completion of the physical design of a work practice, the CE publishes it in the form of a collaborative application for public use.

2.2. Running a session

At run-time, practitioners turn to a library of collaborative applications to select and instantiate an
application as a shared synchronous group workspace. Participants who are new to the work practice may turn to documentation and tutorials of the application to learn of their role and expectations within a group activity. The group incrementally moves through activities and screens to accomplish common goals as designed by the collaboration engineer. As a result, the application provides an end-to-end support for a given group task.

3. Universal data model (UDM)

Typically, the data needs of one collaboration problem often do not correspond to the data needs of other collaboration problems. In order to use the same system for different collaborative problem domains, facilitators often need to change the data model to use data types and relationships that are semantically meaningful in a given problem’s context. This is not easily accomplished because most applications do not provide a dynamically configurable data model. Since facilitators do not always possess the expertise of re-designing an application’s data model, they often have to turn to system developers for an extension or adaptation of the system to the new data needs, which may adversely affect time, budget or quality constraints of the facilitator. A more robust approach to this problem is to let the facilitators construct new models without a re-design of a system’s internals.

Following the basic principles of Entity-Relational modeling, we build our model around Entities that describe concepts and Relationships among concepts. Both of these generic objects have a type property defining the semantic meaning of a concept and its relationships. In addition, an Attribute object contains a key-value pair to provide further description of an associated entity. More formally, properties of the objects are described by the following records:

- Entity: {id, type}
- Attribute: {id, key, value, entityId}
- Relationship: {id, type, superiorId, subordinateId}

An entity can have zero or more attributes; an attribute must have exactly one corresponding entity. Also, an entity can have zero or more relationships; a relationship must have exactly one superior and one subordinate entity.

The primary benefit of the model is that it is generic in the kinds of data that can be captured. Since the model can be populated with any data from any domain, the model is domain independent. A secondary benefit of the model emerges from its generic nature: upon construction of the database schema for the model, the same schema can be reused over and over without the need to redesign for the needs of a particular problem. For example, the tables can store Unit entities that are subordinate to Department entities with MemberOf relationships as well as Aspect entities that are subordinate to Requirement entities with Affects relationships or Order entities that are subordinate to Customer entities with MadeBy relationships. This gives facilitators the flexibility of building complex data models without having to depend on or possess a database analyst’s skills and without the need to define new tables.

4. Population rules

To support construction of dynamic models in the field, the system needs to provide a CE the means for their specification. One of such specification formalisms is population rules, which specify the types of data and the types of relationships between the data. They are written in the format of

SubordinateEntityType-RelationshipType-SuperiorEntityType

Intuitively, a rule defines a graph with two nodes and an arrow between them. A collection of these rules can incrementally declare a data model of any size.

Example: Suppose a CE needs to design a risk management brainstorming session for identification of risks affecting business units of an organization and the means of controlling the risks. One of the steps in the design process is to build a corresponding data model. A succinct declaration of the model with population rules is

Risk-Affects-BusinessUnit
Control-Mitigates-Risk

If, at a later time, the CE decides to separate this data into its own dataset, then the following rule can be added to extend the model:

BusinessUnit-MemberOf-RiskControlDataset

Example: Computer-Assisted Collaboration Engineering (CACE) tool [20] used by a CE may employ the following process flow. Every work practice is packaged as a project, which consists of phases and activities, where activities may in turn be grouped into phases. For every role and every activity there exists a screen, which is displayed to group participants as they move through activities at runtime in a Collaboration Support System (CSS) [20].

All of the objects of this process flow can be represented in the universal data model (UDM) with the following population rules:
1. Phase-childof-Project
2. Activity-childof-Project
3. Activity-childof-Phase
4. Role-childof-Project
5. Screen-childof-Role
6. Screen-childof-Activity

Relationships among entities are directed to enable specification of data hierarchies. This does not prevent declaration of bidirectional relationships, where both entities may depend on each other: e.g. Unit-SubordinateOf-Department Department-SubordinateOf-Unit

From a CE’s user interface perspective, population rules can be entered into the system using groups of text fields – either one for each component.
of a rule or one for each rule (e.g. Figure 1). The CE can then configure screen components to be populated with type-decorated inputs from group participants. For example, a form to accept participant’s contributions can be combined with a drop-down selection list to indicate the type of a contribution (see Figure 1).

The combined benefit of the generic data model and the population rules is the ability to build, at runtime (i.e. in the field), complex data models with data types and relationships that are semantically meaningful within the context of a particular collaborative task. In addition, the same tables can be used to store data generated by an application at runtime as well as the data of the application itself (for instance its process flow as in the example above).

5. Dynamic contribution channels

A synchronous groupware system needs to support group-enabled editing of artifacts. In the same shared environment, for example, one user could be editing a contribution while another moves it higher in a list, another moves it lower, another adds subordinate contributions to it, another links it to other contributions, another breaks links between it and other contributions, another creates a pointer to it so it will appear multiple times in the same context, and yet another deletes it. In groupware applications, these conditions are the expected behavior, not the exception.

One of the primary functionalities in such environment is the ability to observe changes made by other users. While it is possible to broadcast a certain event to all users within a system to notify them of new content, not every user is interested in all events, which wastes system and user resources. A more efficient approach is to let users subscribe for updates to only those artifacts, in which they are interested in. Then, a change to an artifact’s content will be multi-casted to a subset of users interested in that change. This behavior is an example of the observer design pattern [25] of the ‘publish-subscribe’ messaging paradigm [24].

We supplement the observer pattern by defining the criteria for content, which publish-subscribe requests can ask for. The criteria use standardized communication channels. Channel names follow URI-like naming scheme and describe the data contained in a channel’s messages by identifying

1. data kinds: whether a message pertains to a contribution (entity), a contribution property (entity attribute), or a relationship;
2. GUID identifiers for unique determination of event data, and
3. event type: whether it is an add, update or delete event.

For example, the channel name for messages related to updates to a contribution is /contribution/<contributionId>/update. Other examples include /relationship/<relationshipId>/delete and /contribution/<contributionId>/property/<propertyKey>/delete.

We use the ‘push’ model so that the messages communicated over the channels contain the complete state of entities, attributes and relationships to enable recipients construct new or deconstruct existing objects based solely on the message contents. This is more efficient than the ‘pull’ model because the subscribed observers do not need to contact the server after the notification to inquire about further details of an event.

As a result, the combination of dynamic contribution channels and the observer design pattern allows the system to propagate any data changes and provide consistent shared view of data for all users and across all screen components.

6. Implementation aspects

Having discussed the conceptual contributions to the design of groupware, let us now turn to implementation details to observe how these elements are used in practice to provide an ARCADE.

6.1. UDM-based data store

Groupware systems are highly interactive systems, where many users often manipulate the same data objects at the same time. A lot of effort has been spent on researching and defining suitable concurrency control approaches for synchronous groupware. These approaches range from employing a social protocol [27] among the collaborating users, over pessimistic locking approaches or adaptable approaches [28] up to sophisticated approaches for operational transformation [29]. Despite this variety of existing approaches, one of the primary methods for dealing with concurrency in groupware is still the delegation of its handling to a database management system (DBMS). A DBMS guarantees integrity of data by serializing incoming requests into database transactions and committing transactions with no conflicts or rolling those back otherwise [29]. In addition, a DBMS provides data locking mechanisms at a row or table level, which facilitates reduction of
conflicts in highly interactive groupware sessions. Hence, the architecture of most groupware systems includes a data store module that employs a DBMS.

For integration of relational data view of a DBMS and object-oriented view of a groupware system’s implementation language, we use the Active Record design pattern [30], where a table is mapped to a class, a row in the table is mapped to an object of the class, and a column in the table is mapped to the object’s attribute. Active records provide data access functions to read and write to the database tables as well as the domain logic to manipulate raw table data. Since our groupware system’s implementation language is Java, we use Hibernate – a popular implementation of active records for Java.

Figure 2. Universal Data Model (UDM) capable of storing arbitrary relational data. Workspaces are used for conceptual separation of data from different work practices. User credentials are stored separately to provide stand-alone user administration and system-wide access control.

Synchronous multi-user data modification presents race hazard problems, where the data may end up in an unexpected state depending on the timing of read and write requests. For example, when two users synchronously modify a data element, one user’s updates may be overwritten by the other user’s update. We use enterprise-class Spring transaction management libraries to ensure Atomicity, Consistency, Isolation and Durability (ACID) of data [33]. In particular, all external modification of active records initiated by user interface components is conducted through transactional service calls. As a result, the integrity of data is guaranteed irrespective of the number of synchronously editing users.

Figure 2 provides a skeletal view of the UDM implementation in the form of an ER diagram. The core tables of Entity, Attribute, and Relationship are supplemented by the User table to store user credentials in the system and the Workspace table to provide conceptual separation of data from different work practices. Besides the core properties, the tables include modifiedBy, lockedBy and timestamp properties. The modifiedBy property is used for change-tracking of the data based on who modified the data. The lockedBy property is used to edit-lock entities and their attributes to provide single-user editing. The timestamp property is used for version control to indicate the time of the latest update to the entity. Whenever an entity is about to change, the state of the old entity is saved in a history table (not shown) for that type of entity with the ‘from’ and ‘to’ timestamps indicating the period during which the state was active. A sequence of such recorded states represents the entire version history of an entity.

6.2. Collaboration support system clients

One of the major barriers for wider adoption of groupware in the workplace is the usability of collaborative tools. Groupware clients’ installation, their configuration and group member training are labor-intensive tasks that collectively present a steep learning and adoption curve. One of the effective strategies for diffusion of technology is to present it in the form of web-based applications. Since virtually all modern workstations provide web browsers with some Internet connectivity, this significantly reduces any installation and configuration issues and minimizes the training time. In addition, online collaboration for geographically dispersed teams is delivered in web applications by default.

A web-based groupware application needs to support highly interactive activities, which may dynamically change web page contents. JavaScript is a leading scripting language for building of dynamic user interfaces. Besides the existence of many libraries to build web widgets, JavaScript is supported in most web browsers by default and does not require installation of any plug-ins.

Groupware applications typically support groups of tens, if not hundreds, of synchronous users. The
server(s) need to be able to support group interactions with minimal delays. A promising web technology that minimizes the number of web requests/connections and loads the data as soon as it becomes available is Comet [34], which is an asynchronous JavaScript method of pushing new data to web clients without the clients needing to periodically request for new content: i.e. Reverse Ajax. Because of the dynamically updated and asynchronously delivered content, the entire collaborative application can be hosted on a single URL address. New users can type in the URL, provide their login credentials or self-register and join a collaborative session. After joining the session, a user does not need to interact with browser controls such as refresh or navigation buttons, which further minimizes the required user proficiency with browsers.

Group activities need to provide a consistent shared view of data for all users so that one user’s change is propagated to everybody else. We put the concepts of the Observer design pattern and dynamic contributions channels into practice by implementing them in JavaScript with the help of Comet libraries. In particular, construction of a UI component on a user’s screen is followed by a subscription to a dynamic contribution channel for the data contained in the component. The subscription also indicates how messages arriving on that channel should be handled. Then, when one user interacts with the component to submit new data to the server, the server publishes the new data on the contribution channel. Every UI component subscribed to the channel sees the new data and invokes the message handler. As a result, this provides a live update of every user’s browser view in response to any user’s changes.

6.3. Middleware service functions

Having considered the data and user interface modules of collaborative applications, let us now turn to middleware functionality that transfers data between the two modules.

6.3.1. Event notification. The entity persistence layer – Hibernate – provides default event listeners for create, update and delete operations on persisted entities. Overriding these listeners enables custom handling of the events. For example, if a modified entity contains a time-stamp property, then a new record with its state can be saved in a history table to provide a capability for version replay of the entity from the time it was created to the latest modified time-stamp. A more important feature enabled by the listeners is to publish the new state of a modified entity on the dynamic contribution channels. Hence, modification of an entity in the database triggers notification of all subscribed observers about the new state of the entity.

6.3.2. User management. Groupware systems need to authenticate users and enable/disable access to certain collaborative sessions. System access and session access are controlled by distinct techniques and are stored in system and workspace role tables (omitted from Figure 2 for brevity). For system access, a user can be associated with one of four roles: administrator, engineer, facilitator and participant. The engineer role allows a user access to the CACE tool to create and modify collaborative sessions. The facilitator role enables one to acquire and manage (licensed) collaborative applications. The participant role is the minimal access role allowing a user to enter collaborative sessions. Administrators can modify system settings as well as add or modify other user’s accounts.

For session access, a user can be associated with workspace roles that are defined by a CE during the design of a collaborative application. Here, the system provides a configurable component that can be placed on a facilitator’s screen by the CE. The component lists system-registered users on the left and session roles on the right. Then, during collaborative session (at run-time), a session leader can drag-and-drop a username into a session role to give the associated user the specified role in the workspace of the session. A user without a session role cannot enter that session’s workspace.

6.3.3. System settings. Besides user administration, system settings control the content of elements of a collaborative application. Among these elements are projects, phases, activities, roles, screens and screen components. While every application will contain one or more of each of these elements, their configuration will differ from one application to another. This configuration is defined by a CE at design-time and specifies the look-and-feel of an element (e.g. icons, font sizes, font styles) or the documentation of an element (e.g. name, description of a project, duration of an activity, whether a role is a leader role etc.). The content and the form of an element is specified by the system administrator by the means of an XML file that defines the name of a configurable item such as name or icon and the means of its configuration such as input field (single- or multi-line text field) or selection field (radio- or checkbox-button, single- or multi-select drop-down list). This provides the CE complete control over usability aspects of an application. For example, elements’ icons can be customized for the expected audience, or
documentation of activities and roles can provide self-contained instructions for novice users.

6.3.4. UDM data manipulation. Since a UDM-based database can store arbitrary relational data, it is important to provide general-purpose methods for data manipulation. A reasonably general abstraction toward this end is to view the data as a graph, where entities represent its nodes and relationships represent its edges. Every node has a collection of attributes that describe its properties. The edges are directed such that heads are represented by superior entities and tails are represented by subordinate entities.

Given such graph-based view of the UDM, the following basic operations do not depend on specific data domain and can be used by any UDM-based application:

1. Create a new node.
2. Read a node based on node id.
3. Update a node by updating its properties.
4. Delete a node or deep-delete a node and its subordinates.
5. Add relationship between two nodes.
6. Remove relationship.
7. Reorder subordinates by updating the sequence of their id’s in the superior node’s property.
8. Move a node from one superior node to another.
9. Copy a node or deep-copy a node and its subordinates by making it subordinate to a ‘clipboard’ node.
10. Cut a node or deep-cut a node by copying it and also making it subordinate to a ‘recycle bin’ node.
11. Paste a node from the ‘clipboard’ by making a deep-copy of its contents and making the copy a subordinate of the paste target.
12. Undelete a node by removing the relationship to ‘recycle bin’ and restoring the previously deleted relationship.

In addition to these generic operations, UDM-based collaborative applications require workspace and user role management operations: e.g. add/remove workspaces and add/remove roles in a workspace.

Each operation is implemented by a transactional service method. Screen components manipulate server data by invoking service calls. Any change in the server data triggers event notifications to update every user’s data view. Further data changes are based on the updated view. Hence, integrity and consistency of data is maintained for all users.

7. Validation

The prototype of the groupware system is implemented as an open-source (GNU LGPL-licensed) Java-JavaScript web application available at [36]. The API and related documentation can be downloaded from [35].

To validate the capabilities of the groupware server towards the demands of real-world groupware applications, we have subjected the package to a collection of load tests. The tests check functionality of basic Create, Read, Update, Delete (CRUD) functions provided by the Data Access Object (DAO) interface of UDM records. Since CRUD operations can be used as building blocks for any other operation on the data store, correctness and robustness of these functions ensures the same qualities for all other server operations.

To implement load testing, JUnit tests are used. They check correctness of a single data store function. They are executed in parallel to simulate multiple users accessing the data store concurrently. For accuracy and reusability, existing open-source load-testing tools JUnitPerf [31] and JUnitBenchmarks [32] are used. JUnitPerf implements a multi-threaded execution of JUnit tests. JUnitBenchmarks allows for a repeated execution of a JUnit test to collect a reliable estimate of an average execution time of a single test.

The load tests check the average response time of a data store function based on a given number of concurrent users. In particular, a single load test consists of X number of users submitting a request of the same type at the same time. Each load test is executed 15 times, where the first 5 ‘warm-up’ runs are discarded, and the last 10 runs are averaged to arrive at a response time for that operation.

The results of the load tests are illustrated in Figure 3. Load-testing was performed on a notebook machine equipped with Intel Core 2 Duo 2.2GHz CPU, 4GB RAM, Windows Vista 32-bit OS, MySQL 5.1.37 DBMS, and Java SE 1.6.0_18 JVM.

Higher values for update and delete operations are explained by additions of history records for version control. Note that an exponential increase in the number of users produces similarly growing response time indicating that the response time is a linear function of the number of concurrent users.

Field data suggest that a typical collaborative session consists of 20 participants with average contribution rate of 1 per minute and a response time threshold of 3 seconds, after which a groupware system is perceived to be slow or unresponsive. Our load tests demonstrate a throughput of 100 requests per second or 6000 requests per minute. Assuming the typical group size of 20 users, the UDM-based
data store can serve 300 groups in the field concurrently.

![Figure 3. Load-testing the UDM-based server. Rounded response time (in seconds) of each CRUD operation based on a given number of users. Since response time is a linear function in the number of users, the data store can support up to 300 20-person groups under typical conditions in the field.](image)

We interviewed the commercial vendors of three leading GSS products and they reported 3-3.5 person\-years to develop collaborative components and tools (roughly 6000-7000 person\-hours) [20]. Using a PC prototype of a CACE/CSS-based groupware system called Cognito and working with practitioners in industry, we configured six CSS applications ranging from software requirements negotiation to marketing focus groups and general facilitation. The time taken to build CSS applications was $10^6$ times faster than the conventional GSS development practices.

Our prototype advances the CACE/CSS approach by extending the previous state-of-the-art with dynamic data model capabilities among other developments. Load tests reported above (Figure 3) complement the functional results/comparisons and provide additional feasibility evidence for a generic large-scale groupware system capable of supporting end-to-end collaborative work practices.

8. Conclusions

This paper advances groupware technology to overcome several limitations that block widespread use of groupware technology by practitioners who do not have the support of professional facilitators. Following the facilitator-in-a-box strategy, the groupware system’s architecture consists of a CACE tool to develop collaborative ActionCenters and a CSS to execute the ActionCenters. The CACE tool provides a CE complete control in the design of an end-to-end work practice. The UDM together with population rules enables flexibility and generality in handling dynamic data models of a work practice. All system components use graph-based data view to manipulate arbitrary data relationships and dynamic contribution channels provide live update of any modified data. As a result the system is capable of handling arbitrary data in arbitrary relationships in a multi-user environment.

Using a CACE tool, collaboration engineers have demonstrated they can create functionally equivalent collaborative applications in a fraction of the time that conventional software development methods would require. In addition, the resulting applications provide linearly scalable performance for typical conditions in the field.

From the organizational viewpoint, the approach presented here represents substantial resource savings, because organizations can hire a collaboration engineer just once to design and deploy an application that can be used multiple times. In other words, there is no longer a need to hire and retain a professional facilitator to conduct recurring work practices.

However, this outcome does not mean that professional facilitators will not be needed at all. Facilitators will still be required for ad-hoc and high-risk tasks. Further, only a trained facilitator with collaboration engineering skills will be able to successfully design and deploy ActionCenters. Rather than limiting the role of the facilitator, therefore, the ARCADE system lifts and scales the facilitator’s role from direct group facilitation into designing, piloting, training, and maintaining work practices executed by others. With ARCADE, facilitators will be able to create a legacy that lasts beyond the time they spend face to face with a group.

9. References


