In: *Proceedings of the Fourth IFCIS Conference on Cooperative Information Systems* (CoopIS'99)
Edinburgh, Scotland, September 2-4, 1999.
Also: MCC Technical Report CMI-022-99, MCC, Austin, TX, 1999

Providing Customized Process and Situation Awareness in the Collaboration Management Infrastructure[*]

Donald Baker, Dimitrios Georgakopoulos, Hans Schuster,
Anthony Cassandra, and Andrzej Cichocki
Microelectronics and Computer Technology Corporation
3500 W. Balcones Center Drive
Austin, Texas 78759
{dbaker, dimitris, schuster, arc, cichocki}@mcc.com

Abstract

Collaboration management involves capturing the collaboration process, coordinating the activities of the participating applications and humans, and/or providing awareness, i.e., information that is highly relevant to a specific role and situation of a process participant. In this paper we propose an awareness provisioning solution that allows customization of the awareness delivered to each process participant. Unlike existing collaboration management technologies (such as workflow and groupware) that provide only a few built-in awareness choices, the proposed awareness solution allows the specification of what information is to be given to what users and at what time. To support this advanced level of awareness, we require the definition of awareness roles and the specification of corresponding awareness descriptions. Awareness roles can be dynamically created and associated with any process scope. Awareness descriptions define what information is to be given to users in an awareness role. Since awareness roles are created or become visible when they are needed, the existence of an awareness role also determines the appropriate time interval during which the information specified in the awareness description can be delivered. This customized awareness provisioning approach minimizes information overloading and allows the combination of process-relevant information with external information as needed by the process participants. The proposed awareness provisioning solution is employed by the Collaboration Management Infrastructure (CMI), a federated system for collaboration process management. Throughout the paper we use examples from the crisis management domain.

[*]This work was funded in part by DARPA contract F30602-97-C-214, "Serveillance of Complex Events Using Active Agents."

1. Introduction

Collaboration management involves capturing or modeling collaboration processes, coordinating the activities of applications and human participants, and/or providing awareness by communicating collaboration-related information to participants.

The *Collaboration Management Infrastructure* (CMI) has been developed at MCC to accomplish the following objectives:

- manage collaboration processes,
- provide combined process and situation *awareness*, and
- support processes in virtual enterprises as well as in traditional organizations.

CMI technology development is driven by the requirements of many advanced applications that are not effectively supported by existing workflow and groupware technologies. To address these requirements CMI provides a sophisticated Collaboration Management Model (CMM) and a corresponding component-oriented system that implements the CMM. CMM draws existing primitives from workflow and groupware models and introduces new primitives for previously unsupported collaboration process requirements. CMM consists of a *Core* Model (CORE) and several specialized extensions of it. CORE provides a common set of process model primitives that constitute the basis for all extensions. The CMM extensions include specialized process models designed to support coordination, awareness, and services. The *Coordination Model* (CM) provides additional primitives for coordinating participants and for automating collaboration process enactment. The CM may have to deal with coordination processes that may be partially unknown when they start. The *Awareness Model* (AM) is a CORE extension that captures customized process and situation awareness. The *Service Model* (SM) supports reusable process activities and related resources, service quality, and service agreements, as needed to support collaboration processes in virtual enterprises. CM and the coordination capabilities of CMI are described in [8, 7]. SM and service selection and invocation are discussed in [7].

In this paper, we describe the CMI capabilities for providing *awareness*, i.e. AM, relevant parts of CORE, and the related implementations. We define *awareness* as information that is highly relevant to a specific *role* and *situation* of a process participant. Because a human's attention is a finite resource that must be optimized, awareness information must be digested into a useful form and delivered to exactly the users who need it. If given too little or improperly targeted information, users will act inappropriately or be less effective. With too much information, users must deal with an information overload that adds to their work and masks important information.

Awareness provisioning involves the specification of relevant information, gathering these information from a running system, digesting it into a usable form, and delivering it to the appropriate process participants. Unlike existing collaboration management technologies (such as workflow and groupware) that provide only a few built-in awareness choices, *CMI* allows the customization of awareness via *awareness specifications*. Awareness specifications, which are provided by process/awareness designers, define what information should be directed to what users based on their roles in the process. Awareness specifications consist of *awareness roles* and corresponding *awareness descriptions*. A*wareness descriptions* define the information that is delivered to a user that plays a specific awareness role. Such descriptions are made from event patterns that not only describe the desired constellation of events, but also how the information from those events is to be digested. Awareness roles are referenced in awareness descriptions and they are used in delivering customized awareness to process participants. Awareness roles do not have to be the same roles used for process coordination. Furthermore, they can be dynamically created and associated with any process *scope* or *context*, i.e., any collection of process activities and/or resources. The existence of an awareness role determines the appropriate time interval to deliver the information specified in the awareness description, e.g., when such a role is created or becomes visible.

To provide awareness, CMI introduces several process-oriented enhancements to generic event processing technology. These include process-specific event operators, specialized event operators with built-in categorization for process instances, and event operators that accept process-specific parameters.

The remainder of this paper is organized as follows: In Section 2, we discuss some key requirements for awareness provisioning that are not supported effectively by existing technology and provide a critique of related work from the perspective of supporting such awareness requirements. In the following sections we discuss the corresponding CMI solutions supporting process and situation awareness. In particular, in Section 3 we outline CMI’s Collaboration Management Model (CMM) for capturing collaboration processes. In Section 4, we focus on the CORE of CMM that provides the basis for developing CMI’s Awareness Model (AM). AM is described in detail in Section 5. The CMI system architecture and the AM implementation are outlined in Section 6. The conclusion is in Section 7.

2. Requirements and Related Technologies

Although CMI is a general purpose technology that can support many advanced applications [8, 7], in this paper we motivate our work on awareness by focusing on CMI support for the crisis management domain. Similar awareness requirements also exist in command and control, and telecommunications service provisioning applications. The requirements of collaboration processes from these application domains are discussed further in [8, 7].

The basic objective of a crisis management application is to facilitate the resolution, or at least the mitigation, of a crisis situation. Crisis situations appear in virtually all parts of government and economic life. They range from large scale crises, e.g., natural or economic disasters, military tensions and contentions, and epidemic outbreaks, to highly localized crises, like simple accidents. The principal characteristic of a crisis situation is that it occurs *unexpectedly* and that its *exact course is unknown and unpredictable*.

While the response to an unfolding crisis may have a large degree of unpredictably, an organization that responds to a large number of similar crises will develop regularized procedures and protocols for addressing the range of situations to which the organization must respond. The specifics of these procedures may vary greatly from one crisis response to the next, but the overall structure may have a great deal of regularity. A crisis management application must facilitate the regularization of the crisis response, where appropriate, but be flexible enough to accommodate the variation in the crisis response that can occur. Users who are coordinated by a crisis response application must have the power to make on-the-spot decisions that affect the evolution of the crisis response. As a corollary, the users must have the relevant information at hand, i.e., awareness, so that they can make timely and informed decisions.

As an example of dealing with a crisis, consider an epidemic response. Suppose a group of similar disease reports is discovered in a region of the country. The health organization for that region would start a process responsible for understanding the nature of the disease and containing the outbreak. The process primarily involves information management including interviewing doctors and patients involved, communication with the Center for Disease Control or the World Health Organization, and communication with news agencies and doctors involved in containing the outbreak. While the details of the process are specific to the particular outbreak, the process involves practiced responses that are tailored for the situation.

Figure 1 depicts a possible course of an information gathering process as part of the overall epidemic crisis response. Activities are illustrated by horizontal lines. Some of these activities are always required, while others are optional since they depend on current results and decisions made by the process participants.

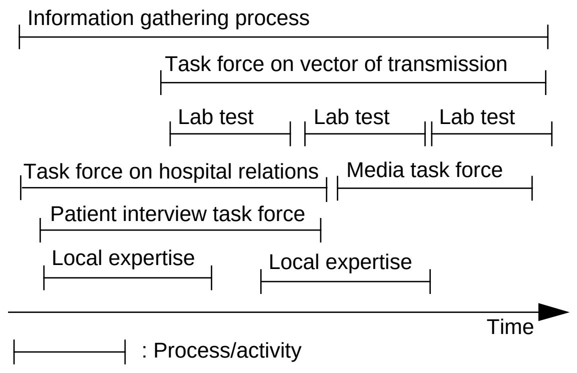
Information gathering process
Task force on vector of transmission
Lab test
Lab test
Lab test
Task force on hospital relations
Media task force
Patient interview task force
Local expertise
Local expertise
Time
: Process/activity

Figure 1. Tasks During Crisis Information Gathering

The process starts when the health agency becomes aware of the outbreak through normal reporting channels. The process ends when the nature of the pathogen is understood and a strategy for containment has been developed (another process would coordinate containment efforts). Depending on the specific crisis situation, the leaders will identify certain areas of interest and create task forces to investigate them. For example, a task force may be formed to contact local hospitals and determine the extent of the outbreak. Another task force may work with those affected to determine likely vector(s) of transmission. The assignment of people is likely to occur after the crisis response process has begun. Depending on the progress of the investigation, task force members may decide to invited external experts or do further lab tests. However, whether of not to issue an additional lab test or acquire additional expertise depends on the collective results of the process. Awareness provisioning is a means of disseminating such information.

In such a crisis response process there are several basic requirements that emerge form the perspective of awareness provisioning. The first basic requirement is that task force members need to be aware of the latest developments that impact their work, such as the results of a lab test or a change in deadlines that affect them. Therefore, to facilitate the effectiveness of the experts involved in a crisis, crisis management technology must filter out irrelevant information and present to each participant only relevant information in a digested form (to further increase information relevance). As an example of this, consider the series of lab tests conducted to assess the impact of the epidemic. Suppose that if any of these tests is positive, the other tests are not necessary. Providing awareness in this case may involve notifying both the test requestor and those conducting the alternative tests when a positive result is found. Such highly relevant awareness provisioning is required to speed the overall process and avoid wasted work.

Another requirement is to effectively determine the specific process participants that must receive each type of digested awareness information. For example, participants in the crisis response process often participate in task forces whose composition is unknown before the process starts. In such dynamically composed inter-organizational teams, members play situational roles in addition to their organizational roles. Situational roles are bound to specific subprocesses, i.e., they are exist only in a specific subprocess scope. Such *scoped roles* cannot be populated a priori; they must be dynamically created and removed as needed by the process. For example, an epidemiologist may be the task force leader. While the epidemiologist role is an organizational role, the task force leader role is a scoped role and may exist only as long as the task force process exists. A task force leader typically requires different awareness than epidemiologists who are simple task force participants.

Finally, in many situations awareness information must be delivered to process participants while they are playing scoped roles. For example, consider again the epidemiologist that plays the task force leader scoped role. If the lab tests have been requested by his task force, then notification of positive results must be directed to its leader. Other epidemiologists, may not need to receive this information.

From the previous discussion, the following awareness requirements emerge:

- Customized *awareness* information is needed to reduce information overloading and increase the relevance of the information provided to the process participants.
- Awareness draws form both process-relevant data and the external world.
- Process participants play scoped roles that may be dynamically defined.
- Awareness information is directed to participants playing scoped roles.

We are not aware of any existing collaboration management technology that addresses these requirements. More specifically, monitoring in *Workflow Management Systems* (WfMS) relates to awareness. Currently, there are standard monitoring APIs available, such as the process monitoring API provided by the Workflow Management Coalition Reference Model [10]. However, unless WfMSs users are willing to develop specialized awareness applications that analyze process monitoring logs, their awareness choices are limited to a few built in options and process-relevant events. In particular, WfMSs currently assume that participants in a process are either “workers” that need to be aware only of the activities assigned to them, or “managers” that must know the status of all the activities in the entire process, i.e., monitor the entire process. Similarly, groupware tools support only limited roles and corresponding awareness that are specific to the intended use of each tool. For example, groupware tools for network presentations, such as neT.120 [4], support “presenter”, “observer”, and/or “hybrid” roles. Presenters are allowed to write on the shared whiteboard and manipulate the application sharing tool, while observers can only observe (read) these resources. Users with hybrid tools can do both of these and they must negotiate and perform coordination outside the scope groupware tools.

Some process oriented systems researchers have used the term *awareness* to describe notifications of specific process activities. Elvin is a general publish/subscribe framework that has been used as part of the wOrlds collaboration system that supports workflows [1]. While Elvin could be considered event-based, subscriptions are done with content-based filtering, but no other form of customized event processing is performed. It is unclear if Elvin is used in direct conjunction with wOrld's workflow enactment events. InConcert WfMS [12] is an example of a process-oriented system with e-mail notification of simple workflow conditions, much in the spirit of this publish/subscribe awareness. While the publish/subscribe model admits that a user will consume the information, these systems provide no mechanism to cater the information for specific roles/classes of users, nor do they address the issue of combining information from multiple sources.

The term “awareness” has also been used in many collaborative systems (not managed by a process specification) primarily to cover only information about one's fellow collaborators and their actions [18, 2, 16]. This limited form of awareness is sometimes called *telepréscence* [9]. One motivation for telepresence is that it allows users to readily determine who is available at remote locations so that ad hoc collaboration may be initiated [6]. However, ad hoc collaboration is less likely in a process-oriented environment where the majority of tasks are coordinated by an explicit process. Our notion of awareness largely subsumes the notion of awareness in collaborative systems both because more than just user information would be considered and because a process model would be leveraged to improve information relevance.

Our concept of awareness follows in the spirit of Dourish who advocates raising the level of abstraction through judicious simplification of the “story a system tells about itself'” [5]. His approach is quite similar to our notion of improving the quality of awareness information through improved contextual relevance. Awareness provisioning in CMI is unique to our knowledge. CMI is the first process-oriented system that can provide highly relevant information tailored to the needs of specific roles process participants play. Furthermore, CMI allows such awareness roles may be dynamically created as needed.

To address the awareness requirements of advanced applications, such as crisis management, CMI combines an advanced processes model with specialized composite event detection technology. These are discussed in detail in the remaining of the paper.

3. Collaboration Management Model (CMM)

CMM is an advanced process-oriented model supported by CMI. It consists of a *Core* Model (CORE) and several specialized extensions of it (Figure 2). The CORE provides a common set of process model primitives that constitute the basis for all extensions. The CMM extensions include models designed specifically to support coordination, awareness, services, and application-specific process models.

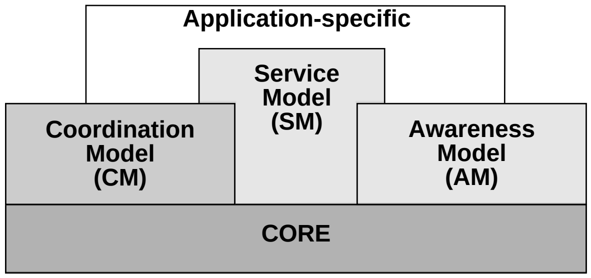

Figure 2. CMM: CORE + Extensions

The Coordination Model (CM) provides primitives for coordinating participants and for automating process enactment. The Awareness Model (AM) is a CORE extension that captures process monitoring and communication of collaboration-related events. The Service Model (SM) supports reusable process activities and related resources, service quality, and service agreements, as needed to support collaboration processes in virtual enterprises. Further extensions can be introduced to support process evaluation and prediction, as well as application-specific process models. Figure 2 indicates this by an application-specific extension atop of CM, SM, and AM.

The CMM is a process *meta model*. An important design decision is whether a process meta model provides a fixed set of modeling primitives or extensibility of primitives via meta-modeling. Meta types for primitives potentially allow more expressive and flexible process models. However, this is typically at the expense of increased complexity. The process models of virtually all COTS WfMSs are examples of models that provide only minimal meta-modeling capabilities. In particular, process models in this category provide meta types only for data resources. The dependency and participant resource types are fixed and there is only a single

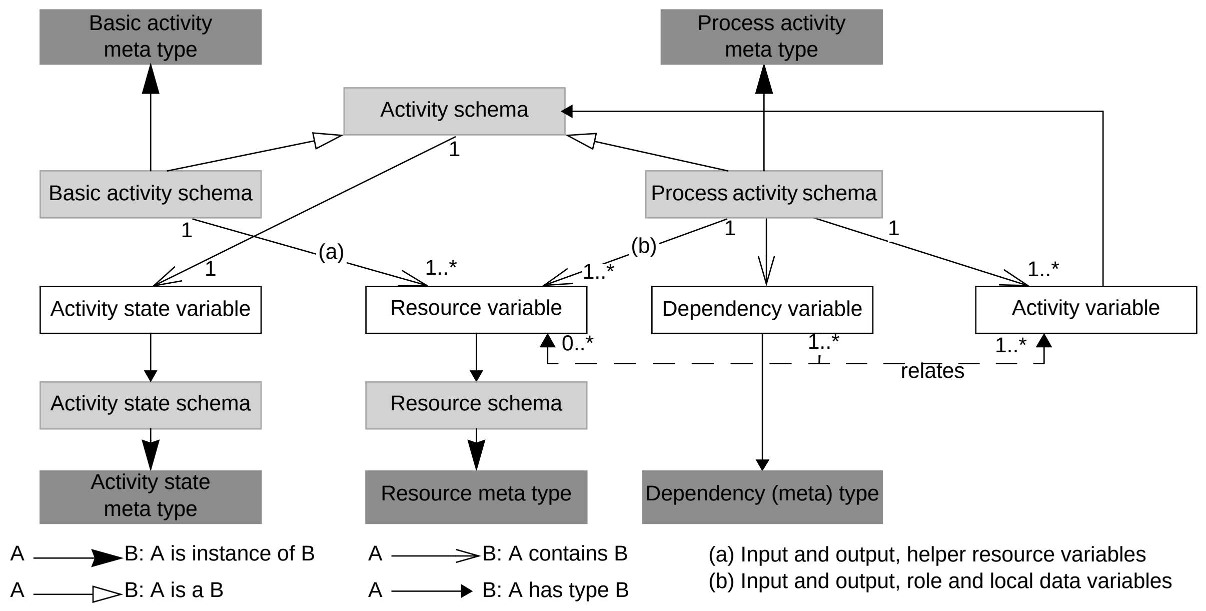

Figure 3. Basic Primitives of the CMM

activity state type. At the other end of the spectrum is the academic *MOBILE* WfMS [13] that supports the broad range of resource and dependency meta types.

CMM is driven by the need to develop a reasonable compromise between the flexibility, expressiveness, and complexity. In particular, as illustrated in Figure 3, CMM provides meta types for activity states (*activity state* meta type) and activities (*basic activity* meta type and *process activity* meta type). The activity state meta type is required to capture application-specific behavior of activities. The meta types for activities can be used to develop application specific process models. For resources and dependencies, CMM follows the approach deployed by COTS WfMSs. It provides resource meta types (*resource* meta type), e.g., to allow for user-defined resource types, and it prescribes a fixed set of available dependency types (*dependency* type).

To allow application modeling, CMM provides *schemas* for activities, activity states, and resources. Schemas are application-specific types that are created from CMM object meta types during process specification. Thus, an application model developed using the CMM comprises of a set of resource, activity state, and process schemas that are instantiated during application execution. Figure 3 shows a high-level view of the basic primitives of the CMM, i.e., activity, activity state, and resource meta types as well as dependency types, and how they are used to define activity and resource schema objects for applications.

Process activity schemas consist of an activity state variable, activity variables, representing the subactivities of a process, resource variables, describing the resources needed during process execution, and dependency variables, defining the coordination rules for the subactivities, e.g., their order of execution. All parts of a process schema are typed. Basic activity schemas are restricted to an activity state variable and a couple of resource variables. Note that the activity and resource variables in Figure 3 are generalizations of the activity and resource primitives in the Workflow Management Coalition (WfMC) reference model [19] and similar primitives used in many commercial WfMSs.

Awareness provisioning is mainly supported by CORE and AM. These are discussed in detail in Sections 4 and 5, respectively.

4. CORE Model

The CORE defines the CMM activity states, including both generic activity states and application-specific extension, and the CMM resources. An important resource type that the CORE provides is *scoped roles*. This is a key resource type in awareness provisioning, since awareness roles are typically scoped roles. Scoped roles are discussed at the end of this section.

Activity states. Each activity schema contains an activity state variable that is associated with an *activity state schema* which determines the possible *activity states* for instances of the respective activity schema and corresponding *state transitions*. A transition from one activity state to another constitutes a (primitive) *activity event*. Events enable CORE to communicate information about activity execution.

Figure 4 shows the *generic* activity state schema defined by the CORE. It is consistent with the proposed standard of the Workflow Management Coalition [20]. Note that a CORE activity state schema enumerates possible activity states and state transitions, but it does *not* define how and when a state transition occurs. CM enhances CORE's activ-

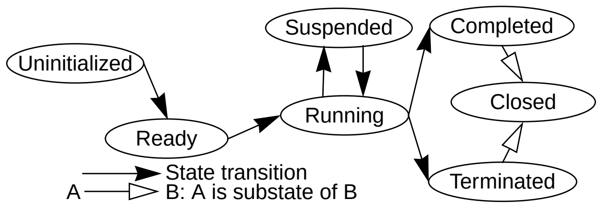

Figure 4. Generic Activity State Schema

ities and activity states with operations that cause state transitions. The CORE's resources are adopted by all submodels without further extensions. CM and SM are outside the scope of this paper. The CM and SM submodels and their implementation issues are discussed in [8, 7].

The generic activity states in Figure 4 capture application independent behavior of activity instances. In addition to the generic activity states, CORE captures *application-specific states* of activities. This allows precise modeling of the application. Other workflow process models, such as METEOR [14] allow for the definition of arbitrary activity states. This can lead to complex process models with activities that do not have common denominator with respect to activity states. Therefore, the CORE's activity state meta model restricts the definition of application-specific activity states to substates of already defined (application-specific) states. This leads to a forest of activity states where the basic activity states are the roots of the trees. A forest of activity states together with the corresponding state transition diagram comprise an activity state schema. Note that state transitions must only connect the leaves of the forest.

Resources. The CORE distinguishes four basic kinds of resource types to be used during an activity execution: data, helper, participant, and context resources. The data, helper, and basic participant resources are similar to those found in many workflow models and WfMS. In particular, the CORE *data* resources correspond to the workflow internal and workflow relevant data in the workflow literature [13, 10]. The *helper* resources are typically programs that provide auxiliary resources to implement basic activities, such as a text editor that is needed for a human to perform a writing activity. Helper resources correspond to invoked applications of the WfMC standard [10].

In addition to the traditional data and helper resources, CORE provides *context* and advanced participant resources. These novel resource types are critical in supporting crisis management and many other advanced applications. The *context resource* is a collection of named resources (similar to a record structure in programming languages). Context resources can be accessed only via context references. This enables the association of a *scope* with any context resource.

Participant resources are either humans or programs. That is, such resources capture actors in the real world that take responsibility to start and perform activities. Both human and program individuals can play (one or multiple) *roles*. Basic participant resources are *organizational roles*. Advanced participant resources are *scoped roles*. Scoped roles are a cornerstone of AM.

Scoped roles. Such advanced roles can be can be dynamically created and exist only within a (context) scope. Unlike usual organizational roles that are global, *scoped roles* are visible only to those activity instances that have access to the enclosing context resource. Scoped roles can be used in the same way within a process specification as usual global roles.

Scoped roles are critical in supporting crisis management applications. Building a task force, for example, may involve selecting an epidemiologists as the task force leader. The task force leader must keep track of the progress of the task force's work. This is not required for epidemiologists that are not task force leaders. Task force-related roles must be dynamically created and assigned to the specific individuals that have been selected to participate in the task force. In general, these roles are independent from the (static) organizational roles of these people, they are only valid inside the task force, and their lifetime is restricted to the one of the task force. Therefore, associating a context with a task force enables the task force-specific roles to be modeled as scoped roles. A similar situation appears in meetings: meeting participants can play a different role during the meeting than their organizational roles, e.g., meeting moderator or presenter. Again, the introduction of a meeting context containing scoped roles provides a solution.

5. AM Awareness Model

The AM is an extension to CORE that can provide timely and highly relevant information to participants. Information in AM is specified and delivered as *awareness* events. Such events include activity state changes, resource status events, and dependency status changes. Furthermore, AM allows for the addition of application-specific event types. Awareness events can be combined into composite awareness events through the use of event operators. Delivery of detected composite events can be directed to users in either global or scoped roles.

In order to motivate the features of the awareness model, we introduce an awareness example that illustrates a specific awareness requirement from the crisis response domain. Suppose that as part of crisis response process, we have a health crisis leader creating a task force to assess the progress of an epidemic in a particular region. This involves the invocation of a process that will coordinate the task force. We will call this process the *task force process*. In addition to specifying the task force members at the beginning of this process, the health crisis leader is also prompted for a deadline for the completion of the task force's work. At any point in the process's lifetime, the leader may change the task force's deadline dues to changes in the external situation. Suppose that the task force process includes an activity that allows task force members to request external information. In particular, assume that *information request* is a subprocess with a separate deadline for delivering the requested information. The information request deadline must be earlier than the task force deadline to allow integration of the requested information into the task force's work. Suppose that the leader changes the task force deadline after an information request has been made. Providing awareness in this situation involves *notifying the information requestor that the task force deadline has been moved earlier than the information request deadline*. Upon receiving this notification, the requestor can renegotiate the request deadline or cancel the request. Without the capabilities of AM, this notification would either be impossible or require significant programming using existing WfMS or groupware technologies. In Section 5.4, we revisit this example in more detail.

In AM, an *event* carries a set of name-value pairs called *event parameters* that give detail about what occurred. Because events are assumed to be self-contained, an event's parameters completely describe the event (e.g., include type, time, and source). This differs from active databases [21] where events may not be self-contained. A *composite event* is an event that is defined to occur as a result of some non-empty collection of events called its *constituent events*. Because events are self-contained, composite events summarize the parameters of the constituent events. Composite events may be constituents of other composite events. Non-composite events, called *primitive events*, come from well defined event producers—either from the enacted process or the outside world.

AM provides an awareness specification language that is used by awareness designers to construct *awareness schemas*. Note that any process participant may be an awareness designer, but this raises serious security issues that are outside the scope of this paper. Therefore, we assume that awareness designers are process designers. Awareness schemas define patterns of composite events, describe how information from constituent events is to be digested, and dictate to whom the result is to be delivered. Formally, an *awareness schema* AS_P on process schema P is defined to be a triplet (AD_P, R_P, RA_P), where AD_P is an *awareness description*, R_P is an *awareness delivery role*, and RA_P is an *awareness role assignment*. AD_P is a composite event specification over event sources visible in the process schema P. Therefore, AD_P specifies awareness information in the form of composite events. R_P is a role visible in the scope of process P that is resolved at composite event detection time to a set of users who are candidates to receive the awareness information specified in AD_P. Finally, RA_P defines what subset of the users in the awareness delivery role will actually receive the information. Together, R_P and RA_P act as delivery instructions for the awareness events detected by AD_P. The awareness description, awareness delivery role, and awareness role assignment are discussed in more detail in Sections 5.1, 5.2, and 5.3, respectively.

5.1. Awareness Description (AD)

The *awareness description* (AD_P) is a composite event specification that has been specialized for the processing of process enactment events a process schema P. A *composite event specification* is a rooted, directed acyclic graph (DAG) where the leaves of the DAG are primitive event producers, the non-leaves are event operator instances, and the edges are connections, i.e., typed *event streams*, between event producers and the consuming slots of event operator instances. An *event operator* is a self-contained, reusable algorithm for recognizing instances of a pattern of constituent events and calculating the parameters of the resulting composite events. An event operator *Eop* may have a type signature $Eop(T_1, T_2, \ldots, T_n) \rightarrow T_{Eop}$, where the operator consumes events from *n* producers with the *i*th source conforming to type T_i and produces events of type T_{Eop}. An instance of an event operator is a running instance of the operator's algorithm which acts as a consumer of multiple typed event streams, called *inputs*, and produces a stream of events called the *output*. An event operator instance can be thought of as a computational pipeline that can produce any number of output events for a single input event.

During execution of the specification, primitive events enter the DAG at their associated leaves and flow to the input slots of operators connected to those leaves. As composite events are generated, they flow to their consumers, which are usually slots of other event operator instances. Composite events that are output from the root of the DAG are said to be composite events *detected* by the composite event specification. The entire composite event specification is an event producer for events produced by its root operator instance.

AM places restrictions on event producers and event operators allowed in awareness descriptions. AM provides a palette of event producers and general operators, however application-specific event producers and operators can be added as needed by the application. Event producers provided by AM are discussed in 5.1.1. The specialization of AM event operators for process support and corresponding operator properties are described in 5.1.2. Finally, the event operators provided by AM are enumerated in 5.1.3.

5.1.1. Event Producers. In this section we discuss two event producers that CMI currently implements: activity state change events and context field change events. Additional event producers are anticipated and AM allows for application-specific event producers.

An *activity state change event* is produced each time a CMI activity changes state. Formally, the primitive event producer $E_{activity}$ has type $T_{activity}$ with the following parameters:

- *time* — the time of the event;
- *activityInstanceId* — the activity instance id of the activity changing state;
- *parentProcessSchemaId* — the process schema id of the activity's parent process, if the activity is not itself a top-level process;
- *parentProcessInstanceId* — the process instance id of the activity's parent process, if the activity is not itself a top-level process;
- *user* — the user responsible for the state change, if any
- *activityVariableId* — the activity variable id of the activity changing state, if the activity is not itself a top-level process;
- *activityProcessSchemaId* — the process schema id of the activity, if the activity is a process;
- *oldState* and *newState* — the old and new states.

Recall from Section 4 that activity states and allowable state changes are defined as part of an activity type's activity state schema.

Section 4 defined a context resource as a named collection of other resources. Contexts are organized into name-value pairs called *fields*. A *context field change event* (or *context event*) is produced each time a field in a context resource is modified. Because of resource scoping in CMM process specifications, a context resource may be associated with several process instances. Formally, the context event producer $E_{context}$ has type $T_{context}$ with the following parameters:

- *time* — the time of the event;
- *contextId* — the id of the context instance;
- *{(processSchemaId, processInstanceId)}* — a set of tuples of process schema ids and process instance ids recording the processes associated with this context;
- *fieldName* — the field name being modified;
- *oldFieldValue* and *newFieldValue* — the old and new values of the field.

AM is open, i.e., it allows for application-specific events to be added to those discusses above. In particular, AM allows the graceful addition of event sources and event operators from outside the process enactment arena. Such event sources may cover events related to information outside the modeled business process or application-specific events from automated systems not directly modeled in the business process. For maximum synergism, external events should be related to the process via application-specific event operators. In the crisis response example, an external event source may be from a news service that has found an article for which a task force has registered an interest, perhaps via an activity that creates a query based on user-supplied keywords. An event from the news service would contain a query id that can be related back to the process instance through an application-specific event operator.

5.1.2. Specialization of AM Event Operators. AM event operators must support the definition of meaningful awareness descriptions that can be authored by a process/awareness designer with minimal effort. To achieve these goals, all AM operators are have been enhanced to directly understand process instances, process nesting, and to work together with ease. In particular, AM operators have the following common properties:

- They output events of a canonical event type.
- They replicate their algorithm per process instance.
- They may be parameterized based on specific features of the process schema to which they are associated.

Canonical Event Type. Nearly all operators take inputs and produce outputs of a canonical event type, denoted C_P, associated with a process schema *P*. The canonical event type simplifies the task of the process/awareness designer because it allows more freedom on how operators can be combined in awareness descriptions and it allows for maximal event operator reuse. The canonical event type has event parameters for the time of the event, the process schema and instance ids, as well as several generic parameters whose meaning depends on the operator that generated it. For example, *intInfo* is a generic information parameter.

Process Instance Replication. Awareness specifications are closely tied to process schemas, but a process schema may have an arbitrary number of instances. Each event operator must therefore replicate its algorithm for each process instance it receives events from. This is necessary so that events are not mixed across process instances. Because the process instance is a parameter on the canonical event type, the operator may simply use that event parameter to access its partitioned internal state. Because all operators in an event description are replicated this way, the entire awareness description is effectively replicated by process instance.

Event Operator Parameterization. AM event operators are actually families of parameterized operators where the parameters are instantiated per operator instance. Parameterized operators are declared as:

$$Eop[p_1, p_2, \ldots, p_m](T_1, T_2, \ldots, T_n) \rightarrow T_{Eop},$$

where the positional event types consumed and the event type produced are as before. The operator is parameterized by *m* operator parameters that must be specified for each instance of the operator. The parameters, which are specified at design-time, customize the behavior of the event processing algorithm embodied in the operator. Usually, the first parameter will be P, the process schema associated with the operator instance's containing awareness description, AD_P. Other parameters are usually constants or items associated with the process schema P, e.g. an activity variable in P. Event operator parameterization increases operator generality with only a small increase in complexity.

5.1.3. AM Event Operator Taxonomy. AM provides *filtering*, *generic*, *count*, *comparison*, and *process invocation event operators*. These five categories of event operators are described below.

Filtering Event Operators. A filter operator takes a primitive event producer as input and outputs some subset of those events as specified by the operator's parameters. Filtering event operators have a one-to-one correspondence with the available primitive event types. AM provides activity and resource filter operators. Additional filter operators, such as for event sources external to a process, can be added as necessary.

For example, the *activity filter* operator is parameterized by a process schema P, an activity variable in that process schema *Av*, a set of old states and a set of new states. In particular, $Filter_{activity}[P, Av, States_{old}, States_{new}](T_{activity}) \rightarrow C_P$ takes the activity state change event type $T_{activity}$ as input, and emits an event of type C_P only when the activity variable in that process makes a state transition from one of the old states to one of the new states. Note that the only source of events of that type is $E_{activity}$, the single source of activity state change events. If the activity occurs in process schema P (*parentProcessSchemaId*), and it is an activity associated with activity variable *Av* (*activityVariableId*), and the old state of the event is in the set $States_{old}$, and the new state of the event is in the set $States_{new}$, then an output event is generated.

The *context filter* operator, $Filter_{context}[P, Cname, Fname](T_{context}) \rightarrow C_P$, is parameterized by a process schema, a context name, and a field name. It takes the primitive context field change primitive event source $T_{context}$ as input and outputs events of type C_P only when there is a value change to the specified field in a context of the specified name associated with the specified process schema. Note that the only source of events of that type is $E_{context}$, the single source of context state change events. If the context event occurs that is associated with process schema P (in *processSchemaIdList*), and the context name matches *Cname*, and the context field name matches *Fname*, then an output event is generated. When appropriate, the new field value is copied to the *intInfo* output event parameter.

As we discussed in 5.1.1 for event sources, AM allows the addition of filtering operators that can be attached to additional primitive event sources as needed. For example, a sentinel filter operator can be added to filter health crisis-related events as needed to support a specific participant role in process for managing a crisis.

Generic Event Operators. Most event processing systems define basic operators for sequence, conjunction, and disjunction. In the following paragraphs, we outline the AM variations of these operators.

The *conjunction* operator, $And[P, copy](C_P, \ldots, C_P) \rightarrow C_P$, takes two or more ($n$) inputs of event type C_P and emits an event of type C_P when an event has been seen on all input slots. More specifically, the operator generates a composite event when some input event, e_i is seen on each input position i, with no constraints on order. The operator parameter *copy* is an integer ($1 \leq copy \leq n$) that selects the input event whose parameters (except time) will be copied to the output composite event.

The *sequence* operator, $Seq[P, copy](C_P, \ldots, C_P) \rightarrow C_P$, takes the same inputs as the *And* operator. The operator generate a composite event when an event has been seen on all input slots in slot order. The *disjunction* operator, $Or[P](C_P, \ldots, C_P) \rightarrow C_P$, takes two or more ($n$) inputs of event type C_P as input and merely echoes every input it receives as its output.

Count Event Operator. The *count* operator maintains a count of the number of input events seen (per process instance) and it emits that value as the *intInfo* parameter on its canonical output event. $Count[P](C_P) \rightarrow C_P$, takes the event type C_P as input and outputs an event for every input seen. The count operator is most useful when combined with the comparison operators, described next.

Comparison Event Operators. The *single input comparison* operator, $Compare1[P, boolFunc1](C_P) \rightarrow C_P$, takes the event type C_P as input. The operator generates a composite event as output when the *intInfo* canonical input event parameter (a generic integer value) satisfies the boolean function, *boolFunc1*. In this case, the parameters of the resulting composite event are copied from the input. If the function is not satisfied, the input event is ignored.

The *double input comparison* operator, $Compare2[P, boolFunc2](C_P, C_P) \rightarrow C_P$, takes two event producers of type C_P as inputs. The operator generates a composite event as output if inputs have occurred in both input positions and the latest *intInfo* canonical input event parameters satisfy the two-parameter boolean function *boolFunc2*. In this case, the parameters of the resulting composite event are copied from the latest input, irrespective of its position.

Process Invocation Event Operator. The *process invocation* event operator is the only operator that allows events associated with one process schema to be translated into events associated with a different process schema. This translation is only meaningful if one process instance invokes the other as a subprocess. The process invocation event operator allows events associated with one process to be translated to the equivalent event relative to its invoking process. (Note that in order to combine events from two process instances that are not directly related through a subactivity invocation, the processing must occur in a common ancestor process, with one process invocation event operator used for every sub-activity invocation involved.) The process invocation event operator, $Translate[P_{invoking}, P_{invoked}, Av](T_{actvity}, C_{Pinvoked}) \rightarrow C_{Pinvoking}$, takes two event producers as input: one of the primitive activity event type and an event producer of the invoked process $C_{Pinvoking}$. The operator parameter *Av* is an activity variable appearing in the process schema $P_{invoking}$ that invokes a sub-activity of process schema $P_{invoked}$. Input events are translated only if an input event is associated with an instance of the process $P_{invoked}$ that was invoked through activity *Av* in the calling process $P_{invoking}$. If this condition is not met, the input event is ignored. (The first event input, $T_{activity}$, is required in order to provide the necessary information for the translation between process instances.)

5.2. Awareness Delivery Role (R)

The *awareness delivery role* (R_P) is a role that indicates the participants of P who shall receive the awareness events specified in the corresponding awareness description. An awareness role may be either a global (organizational) role or a scoped (dynamic) role visible to P. In CMI, awareness delivery roles may differ from the roles used for process coordination, but the same specification mechanisms apply, regardless of usage.

As we discussed in Section 4, scoped roles allow AM to tailor the awareness information for individual process participants as needed to fulfill their responsibilities. For example, in crisis management, scoped roles allow the customization and delivery of awareness to be performed while the process is in progress. We are not aware of any other collaboration management technology that currently provides such awareness capabilities.

5.3. Awareness Role Assignment (RA_P)

The *awareness role assignment* (RA_P) allows a specific subset of the awareness delivery role to actually receive the information from the composite event recognized by the awareness description. The awareness role assignment is an arbitrary function on the set of users gathered by resolving the awareness role that returns a subset of those users. The function may choose users that should receive awareness information based on their load or whether they are currently signed-on to the system. Currently, the only implemented awareness role assignment function is the identity function, implying that all users in the awareness delivery role will receive the information.

5.4. Awareness Schema Example

Consider again the problem of providing awareness of a deadline violation that was introduced at the beginning of Section 5. In modeling these processes, we assume that the task force process creates a context (*TaskForceContext*) which contains the task force's membership (*TaskForceMembers*) and the deadline (*TaskForceDeadline*) as fields. This context would be passed to the *information request* subprocess. The information request process creates its own context (*InfoRequestContext*) containing, among other things, a role for the requestor (*Requestor*) and the information request deadline (*RequestDeadline*). The requestor is the member of the *TaskForceMembers* role who invoked the information request. The requestor role is created explicitly to identify the specific individual that requested the information. This is necessary because there may be more than one individual playing the task force member role. The *Requestor* role is a dynamically created awareness delivery role that identifies the individual who will receive the deadline violation event. Furthermore, the *Requestor* role disappears upon completion of the information request process, i.e., it is a scoped role. The deadline violation awareness specification is as follows: $AS_{InfoRequest}$ = ($AD_{InfoRequest}$, *InfoRequestContext.Requestor,* Identity), where
$AD_{InfoRequest}$ = *Compare2[InfoRequest, <=](op1, op2),* and:

- *op1 = Filter*$_{context}$*[InfoRequest, TaskForceContext, TaskForceDeadline]*$(E_{context})$, which emits an event upon creation or modification of the task force deadline and
- *op2 = Filter*$_{context}$*[InfoRequest, InfoRequestContext, RequestDeadline]*$(E_{context})$, which emits an event upon creation or modification of the information request deadline.

The implementation of this awareness schema is discussed in Section 6.2.

6. Implementing Awareness Provisioning

In the following sections we discuss how CMI implements the awareness provisioning solution that is prescribed by the AM. In Section 6.1, we outline the CMI system architecture. Section 6.2 describes how awareness specifications are created in CMI. A description of the mechanism for gathering primitive events at run-time is provided in Section 6.3. The composition and processing of such event is according to the awareness specifications is discussed in Section 6.4. The resulting awareness information delivery to the appropriate participants is elaborated in Section 6.5.

6.1. CMI System Architecture

The architecture of the CMI prototype is depicted in Figure 5. The CMI system follows a client-server approach with the CMI Enactment System as the server. CMI leverages several COTS (commercial, off-the-shelf) software systems, most notably the commercial WfMS, IBM FlowMark [11]. The CMI Client for Participants is a suite of tools employed at run-time by users coordinated through

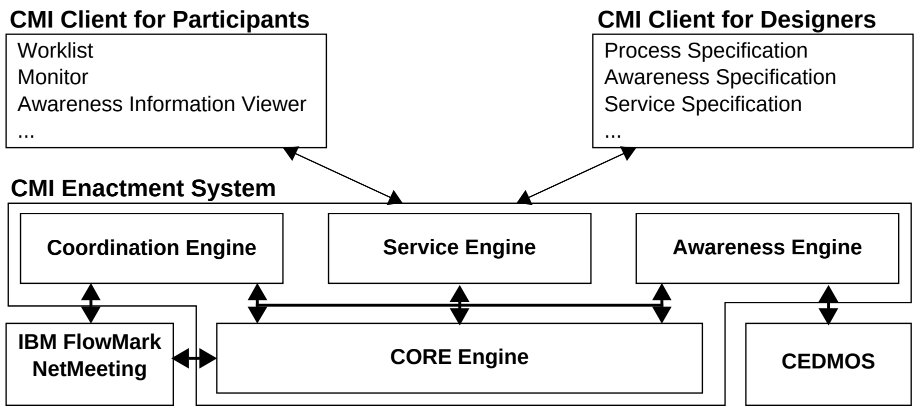

Figure 5. CMI System Run-time Architecture

CMI processes. In CMI parlance, such users are called *participants*. The Client for Participants contains a variant of the traditional WfMS worklist, a process monitoring tool, and a viewer for delivered awareness information. The CMI Client for Designers is a suite of build-time tools that includes the Awareness Specification Tool.

The CMI Enactment System is a collection of communicating agents acting as a single server. The components and their interconnections largely resemble the interrelationships between sub-models in CMM. The Awareness Engine is primarily responsible for implementation of the CMM Awareness Model (AM). However, additional awareness-related functionality appears in both CMI clients.

The CMI awareness engine uses a specialized version of CEDMOS. This is a general event processing system and it is described in [3]. CMI specializations include those described in Section 5.1.2. The CMI Awareness Specification Tool is also a customization of the CEDMOS composite event specification tool.

6.2. Awareness Specification

The CMI graphical awareness specification tool is a build-time client for designers that allows the creation and editing of awareness specifications as defined in Section 5. The awareness specification GUI tool is a composite event specification tool that hides much of the complexity of general composite event specification from the designer. Each window of the tool has a one-to-one association with a previously specified process schema; all awareness schemata associated with that process schema are edited in that window.

Awareness specifications in CMI closely resemble the awareness schemas from AM. Each awareness schema is a rooted, directed acyclic graph (DAG) whose leaves are the primitive event producers, interior nodes are event operators and the root is a special *output event* operator that adds delivery instructions to its input event. This operator has not been previously discussed because it is an artifact of the implementation that simplifies the awareness specification user interface. The output operator's delivery instructions include the awareness delivery role and awareness role assignment, described in Sections 5.2 and 5.3, as well as a user-friendly description of the event. Both interior nodes and leaves may be shared amongst all awareness schemata DAGs in an awareness specification window. The complete set of awareness schemata associated with a process can thus be thought of as a single, multiply rooted DAG.

A designer creates an awareness schema in three steps. First, he places instances of the desired operators (as boxes) in the awareness specification window, which always contains distinct representations (diamonds) for each of the primitive event sources. Second, he specifies the edges of the DAG through a simple direct manipulation mouse interaction. Recall that the edges represent a connection between an event producer and a positional slot of an event consuming event operator instance. Each event operator has constraints on the cardinality and event types permitted for each slot. Third, for operators that allow customization of their behavior through parameterization, the designer can invoke a dialogue-based operator-specific editor from an operator instance, thus allowing him to specify parameters that customize its behavior. A designer can rapidly author complete awareness schemas without specialized knowledge of event processing.

Figure 6 shows a typical awareness specification window in the CMI awareness specification tool. The entire window is associated with one process schema having two awareness schemas, one of which is circled. The boxes are event operators. The special output operator (seen here as Output) encodes the awareness schema’s awareness delivery role and awareness role assignment. The DAG that serves as the input to the output operator is the awareness description. Diamonds represent primitive event sources. The dark lines are event connections between event producers and their consumers. The awareness schema on the right hand side is

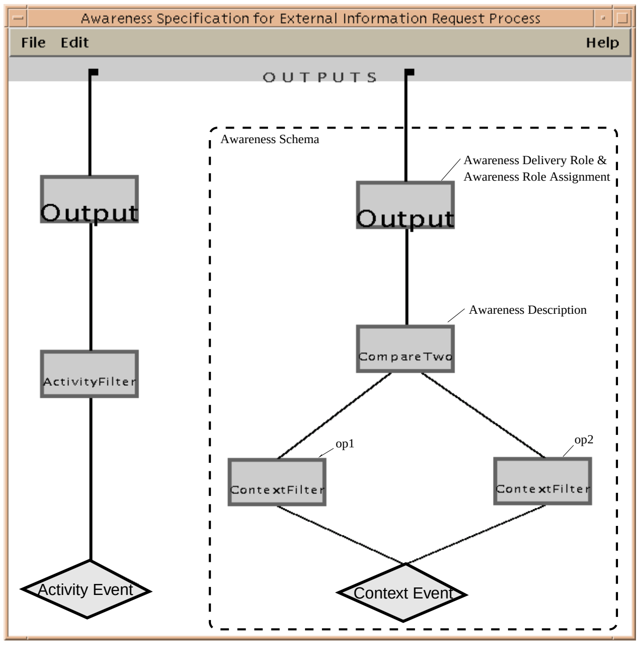

Figure 6. The CMI Awareness Specification Tool

that of our example from Section 5.4. The constituent event operators *op1* and *op2* are annotated for clarity.

6.3. Primitive Event Gathering

There are two main issues with primitive event gathering in the AM implementation: (1) the kinds of primitive events and (2) the mechanism to transmit them from their source to the Awareness Engine for processing. Primitive event sources in CMI consist of activity state change events and context resource field change events described in Section 5.1.1. Depending on the source of the primitive event, a different software component may require instrumentation to gather it. Activity state change events are gathered at the Coordination Engine, for example, and context resource field assignments are gathered from the CORE Engine.

The implementation of AM provides *event source agent*s for gathering primitive events and delivering them to interested software components. Conceptually, the event source agents in CMI are part of the Awareness Engine, though they are tightly bound to the actual event sources.

6.4. Composite Event Processing

At build-time, the designer-specified awareness schemata are automatically transformed into one or more *detector agents* that embody one or more awareness schemas. The resulting agents become part of the Awareness Engine. The agent(s) consume primitive events, perform the event processing, and send recognized composite events, complete with delivery instructions, to the awareness delivery component, described next.

6.5. Awareness Delivery

Delivery of awareness information to the targeted participants is orchestrated through two CMI components: the *awareness delivery agent*, which is part of the Awareness Engine, and the *awareness information viewer*, which is part of the CMI Client for Participants. The awareness delivery agent consumes all composite events of the type produced by the special output operator that was added in the awareness provisioning implementation. Recall that the output event operator adds delivery instructions to its input event. When the agent receives such an event, it resolves the awareness delivery role and awareness role assignment from the event's delivery instructions to a set of participants through an interaction with the CORE Engine. The information from the event is then queued for each participant in the set. A persistent queue is necessary because a participant is not assumed to be logged-on to the system when he receives an awareness event. The awareness information viewer in the CMI Client for Participants is responsible for registering an interest in the event queue for its user, retrieving event information, and displaying it to him. Issues of event aggregation, priority, notification mechanisms, and follow-on actions are under further consideration.

7. Conclusion

Existing technology for collaboration management typically offers only a few built-in choices for providing information to the humans and applications participating in a collaboration process. These typically include some discrete choices such as the list of the process activities a participant has to perform (worklist items), information about the status of specific shared process resources (document or whiteboard status), or information about every activity and resource in the process (monitor data). In many advanced applications these built-in choices either overload participants with information, or the participants have to use complementary tools to gather or communicate information not provided. The main contribution of this paper is the ability to tailor the awareness information for individual process participants as needed to fulfill their responsibilities. Furthermore, in applications that involve dynamic change, such as crisis management, the proposed technology allows the customization and delivery of awareness to be performed while the process is in progress. This is accomplished by supporting (dynamic) context-specific roles. We are not aware of any other collaboration management technology that currently provides such awareness capabilities.

The CMI system has been successfully used in a DARPA-funded demonstration in the intelligence gathering domain. The demonstration involved the specification of nine collaboration processes with more than fifty CMM activities. Some of these processes are open-ended, i.e., they may last anywhere from 15 minutes to several weeks. CMM activity translation into the commercial WfMS used by the CMI system resulted into a few hundreds of WfMS activities. In addition we developed eight awareness specifications and thirty basic activity scripts for creating and managing context resources. We discovered no CMM limitations in capturing the crisis management processes. The CMI system provided all required functionality for supporting the specified crisis management processes. In most cases, the worklist, awareness, monitoring, and hybrid tools provided sufficient GUIs for coordinating the processes participants, and provided functional awareness.

References

[1] Bogia, D.; Kaplan, S.M.: Flexibility and Control for Dynamic Workflows in the wOrlds Environment. In *ACM Conf. on Organizational Computing Systems*, 1995. Available via http://www.dstc.edu.au/worlds/Papers/abstracts.html.

[2] Brinck, T.; McDaniel S.E.: CHI 97 Workshop on Awareness in Collaborative Systems. In *Proc. of CHI'97: Human Factors in Computing Systems*, 1997. Position papers available via http://www.usabilityfirst.com/groupware/awareness/.

[3] Cassandra, A.R.; Baker, D.; Rashid, M.: CEDMOS: Complex Event Detection and Monitoring System. *MCC Technical Report CEDMOS-002-99*, MCC, Austin, TX, 1999.

[4] DataBeam: *neT.120*. http://www.databeam.com/net120/, 1999.

[5] Dourish, P.; Accounting for System Behaviour: Representation, Reflection, and Resourceful Action. In *Proc. of Computers in Context (CIC'95)*, 1995.

[6] Dourish, P.; Bly S.: Portholes: Supporting Awareness in a Distributed Work Group. In *CHI'92 Conf. Proc.: Human Factors in Computer Systems*, 1992.

[7] Georgakopoulos, D.; Schuster, H.; Cichocki, A.; Baker, D.: Managing Process and Service Fusion in Virtual Enterprises. *MCC Technical Report CMI-015-99*, MCC, Austin, TX, 1999.

[8] Georgakopoulos, D. ; Schuster, H. ; Cichocki, A.; Baker, D. : Collaboration Management Infrastructure in Crisis Response Situations. *MCC Technical Report CMI-009-99*, MCC, Austin, TX, 1999.

[9] Gutwin, C.; Greenberg, S.; Roseman, M.: Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. In *People and Computers XI (Proc. of HCI'96)*, 1996.

[10] Hollingsworth, D.: *Workflow Management Coalition The Workflow Reference Model*. Workflow Management Coalition, Document Number TC00-1003, Issue 1.1, 1995. Available via http://www.wfmc.org.

[11] *IBM FlowMark - Managing Your Workflow*. IBM, http://www.software.ibm.com/ad/flowmark/, 1996.

[12] InConcert: *InConcert*. http://www.inconcertsw.com/welcome.htm, 1999.

[13] Jablonski, S. ; Bußler, C.: *Workflow Management - Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.

[14] Krishnakumar, N.; Sheth, A.: Managing Heterogeneous Multi-System Tasks to Support Enterprise-Wide Operations. In: *Distributed and Parallel Databases*, An International Journal, Vol. 3, No. 2, 1995.

[15] Microsoft: *NetMeeting*. http://www.microsoft.com/netmeeting/, 1999.

[16] Pedersen, E.R.; Sokoler, T.: AROMA: Abstract Representation of Presence Supporting Mutual Awareness. In *Conf. Proc. Human Factors in Computing Systems (CHI'97)*, 1997.

[17] Segall, B.; Arnold, D.: Elvin Has Left the Building: A Publish/Subscribe Notification Service with Quenching. In *Proc. of the 1997 Australian UNIX and Open Systems Users Group Conference (AUUG'97)*, 1997. Available via http://www.dstc.edu.au/Elvin/.

[18] Sohlenkamp, M.; Chwelos, G.: Integrating Communication, Cooperation, and Awareness the DIVA Virtual Office Environment, In *Proc. of the Conf. on Computer Supported Cooperative Work (CSCW'94)*, 1994.

[19] Workflow Management Coalition: *Interface 1: Process Definition Interchange Process Model*. Document Number WfMC TC-1016-P, Version 7.04, November 1998. Available via http://www.wfmc.org.

[20] Workflow Management Coalition: *Workflow Management Application Programming Interface (Interface 2&3) Specification*. Document Number WfMC-TC-1009, Version 2.0e, July 1998. Available via http://www.wfmc.org.

[21] Zimmer, D.; Unland, R,.: The Formal Foundation of the Semantics of Complex Events in Active Database Management Systems. *Technical Report 22/1997*, C-LAB, Paderborn Germany, 1997. Available at http://www.c-lab.de/~aatools/external/1997/cr97_22.ps.gz.