The Pennsylvania State University
The Graduate School
College of Information Sciences and Technology

DESIGNING FOR AWARENESS PROMOTION IN DISTRIBUTED, COMPLEX COLLABORATIVE ACTIVITIES

A Dissertation in
Information Sciences and Technology
by
Bo Yu

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

Spring 2013

The thesis of Bo Yu was reviewed and approved* by the following:

Guoray Cai
Associate Professor of Information Sciences and Technology
Thesis Advisor, Chair of Committee

Alan M. MacEachren
Professor of Geography

Mary Beth Rosson
Professor of Information Sciences and Technology

Xiaolong Zhang
Assistant Professor of Information Sciences and Technology

*Signatures are on file in the Graduate School.

Abstract

One of the major challenges to support complex, distributed geo-collaborative activities is to integrate effective coordination mechanisms to manage different types of work dependencies. This study focuses on event-based awareness mechanisms to support the management of dependencies by giving team members an awareness of what each other are doing or have done so that participants can adjust and coordinate their work in a more flexible way. Unlike existing event-based mechanisms, this study (1) relies on a deep understanding of the variety and dynamics of dependencies in these activities to distribute awareness events, and (2) provides explicit visualization and interaction support for the interpretation of awareness events.

The central idea of the approach to modeling dependencies and awareness events is based on a formal structure of collaborative activities. Rooted in the SharedPlans theory [46], the activity model treats a collaborative activity as an evolving shared plan situated in a set of physical and mental contextual factors. Such a model of collaborative activities then can be used to (1) interpret awareness events according to how they are related to the current activities, and notify the relevant users of the events based on the identification of dependencies between the events and users' current focuses.

Then this study illustrates the visualization and interaction provided by the activity model through a design scenario, where a couple of first responders in an emergency response team can use it to help generate and interpret awareness information to manage dependencies. To investigate the impacts of complexities in geo-collaborative activities on the actors' ability to interpret awareness events, an experiment is being designed and performed. The general hypothesis is that the effectiveness of our visualization and interaction design to support awareness interpretation is correlated to the level of complexity of collaborative activities.

Table of Contents

List of Figures

List of Tables

List of Corrections

Chapter 1

Introduction

1.1 Problem Scope

Support for awareness is one of the most active research areas in computer supported cooperative work (CSCW) [28, 93, 83]. At its essence, awareness refers to the ability of collaborators to understand each others' activities and relate them to a joint context [83]. This context is essential for collaboration as it is used to ensure that individual contributions are relevant to the groups activity as a whole, and allow groups to coordinate the process of collaborative working [28].

One general design concern for awareness systems is that awareness must be achieved with minimal attention and effort from the participants of teamwork [64]. Awareness falls into the category of 'the articulation work' that is required for coordination but not the primary goals for collaboration [92]. Taking extra time and effort to achieve awareness can interrupt the current line of action, and therefore damage team performance [44]. As a result, a large number of awareness mechanisms and tools have been proposed in the literature, aiming at promoting awareness in a relatively effortless way [83].

Although much progress has been made in designing awareness mechanisms to support team activity at relatively small and medium scales [5], it becomes a much more difficult task to promote awareness in complex and highly distributed activities [17]. The geo-collaborative activities of interest in this study are a subset of these complex collaborative setting, where awareness is unlikely to be achieved effortlessly, and hence it becomes even more important to design computational ar-

tifacts that actively promote awareness. In the rest of this section, we first describe the characteristics of collaborative settings that we consider as geo-collaboration in this study, and then present the challenges for supporting awareness in geo-collaboration that motivates our work.

1.1.1 Geo-Collaboration

How we define complexity: 1. distributed in geography, 2. number of activities and dependencies, 3. task-specific asymmetries among actors, 4. level of contingency

The geo-collaborative activities we consider in this paper are a subset of complex collaborative setting, with the following characteristics:

1. Multiple team members are geographically distributed.
2. They are engaged in tightly interdependent activities that require effective coordination.
3. They work in dynamic setting that entails rapid and frequent changes in environment and activities.

Examples of geo-collaboration within these boundaries abound in practical applications such as crisis management, resource management, military exercises, and science explorations. For the ease of illustrating the work we present in this study, we will frequently refer to the following scenario in emergency response.

An Emergency Response Scenario. A chemical factory near an urban area was exploded and caused a major pollution. To respond to this critical incident, task force is formed that includes search and rescue teams, decontamination teams, medical treatment teams, and transportation teams. Teams are dispatched and configured geographically to cover the impacted area. Each team cover a functional area of the overall mission. Search and rescue teams patrol the incident area to search for victims and report their locations and status. Discovered victims are first decontaminated (by one of the Decontamination teams) before they can be moved to other facilities. If a victim is wounded, he or she will be scheduled and transported to a medical station for treatment. All transportation needs for moving

victims to treatment stations and shelters are handled by the transportation team. Although teams are working autonomously on their local tasks, they must coordinate their capacity, schedule, and priority to deal with emerging and unexpected situations in order to save and protect all the victims in an efficient fashion.

Such an emergency situation usually involves multiple individuals and organizations that are distributed in different geographic locations. Figure 1.1 shows a hypothetical distribution of tasks and teams in relation to the disaster area. Because of the distribution, workers structure their tasks so that they can work relatively autonomously within their local environment and responsibilities. In the same time, due to interdependencies among sub-activities [95], they find themselves working on a multitude of different activities, interleaving them in ways that seem best suited for getting their work accomplished given the practical pressures. Furthermore, exceptions to the planned responses are a common and critical factor in these activities [113]. What specific information is of concern and interest to a given individual is changing rapidly.

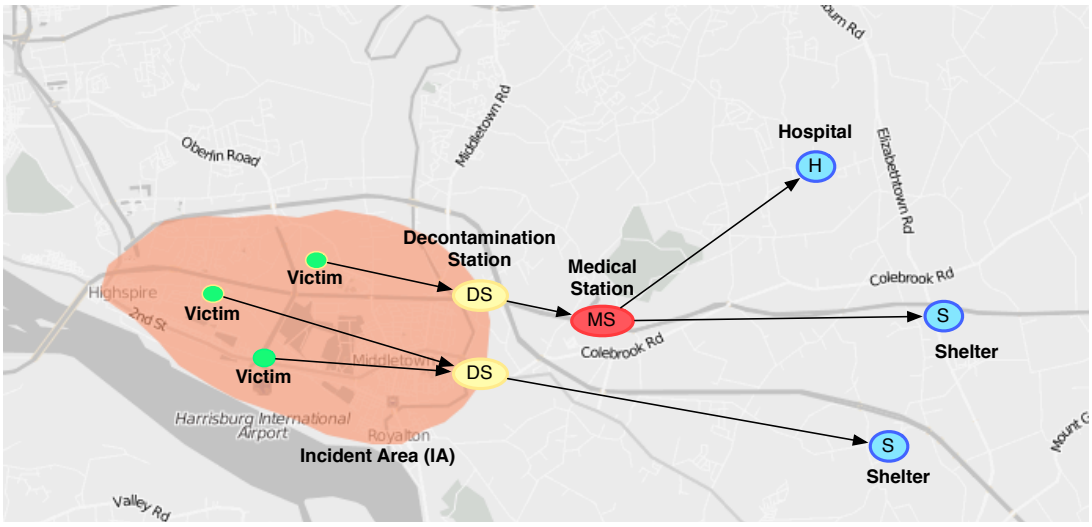


Figure 1.1. An emergency scenario

1.1.2 Challenges in coordinating geo-collaboration

This scenario clearly demonstrates two major challenges to support awareness in complex geo-collaborative activities: the high level complexity and contingency of collaborative activities.

This section needs to be updated to more focus on the awareness support.

1.1.2.1 High level of complexity

With low degrees of complexity, the coordination of cooperative work can be achieved by means of mutual awareness and alignment [93]. As demonstrated by the body of rich empirical studies of cooperative work within CSCW [51], actors tacitly monitor each other; they perform their activities in ways that support coworkers' awareness and understanding of their work; they take each others' past, present and prospective activities into account in planning and conducting their own work; they gesture, talk, write to each other, and so on, and they mesh these interactional modalities dynamically and seamlessly . This appears effortless, because, to a competent member in the flux of doing the work and thus attuned to the changing state of the field of work what the colleagues next to him are doing is immediately meaningful; it does not require interpretation, reflection, contemplation to know why they are doing what they are doing or why they are not doing something else [93].

However, in the complex work settings that characterize modern industrial, service, and administrative organizations where hundreds or thousands of actors engaged in myriads of complexly interdependent activities, the task of coordinating the interdependent and yet distributed activities is of an order of complexity where the mutual awareness mechanisms are far from sufficient. Carstensens study of a software development project [23] clearly illustrates the problem of scaling with complexity. In previous projects the systems they had been constructing had been small and the programming work had been done by a couple of programmers. In these projects they had been able to manage their interdependencies practically effortlessly. They had been working next to each other and had practically unconstrained access to consulting each other and to monitoring each others work. At the time of the study, however, a new project had been undertaken in which

the engineers were building a significantly larger system comprising many hundred thousands lines of code. Their traditional coordinative practices were now quite inadequate. The interdependencies of their cooperative effort now transcended the local practices, and they were faced with situations that the effects of their local activities to other regions of the cooperative effort are not immediately and straightforwardly evident. To deal with the ensuing crisis, the ensemble had to develop a set of formal coordinative artifacts, such as the bug report forms, to regulate local practices.

One of the major challenges demonstrated in the scenario is characterized by the degree of *complexity* of dependencies that can exist in coordination work. With low degrees of complexity (e.g. when the number of dependencies that need to manage is limited, or their effect is only within the scope of local practices), the coordination of collaborative work can be achieved by means of human communicative and cognitive skills [94]. However, in the complex work settings as described in the scenario, multiple dependencies can co-exist at the same time, and they together form a web of dependencies where state changes of one of them can cause chain effects on others. Faced with a high degree of complexity of coordination work, collaborative actors often use a category of symbolic artifacts which, in the context of a set of procedures and conventions, stipulate and mediate coordination work and thereby are instrumental in reducing its complexity and in alleviating human effort [100]. These artifacts, together with the concomitant procedures and conventions are called 'coordination mechanisms' [94]. The first goal of this study aims at building an integrated coordination mechanism that would keep track of the state of work and to manage relations and dependencies among actors, tasks, and resources in order to reduce the complexity of coordination work in geo-collaborative activities.

1.1.2.2 High level of contingency

Developing appropriate coordination mechanisms must be based on a solid understanding of various collaborative dependencies, i.e. the capabilities of coordination mechanisms should match coordination requirements of the task at hand. In relatively static work domains such as process control or manufacturing, the set of work dependencies is largely known in advance and thus it becomes feasible to

specify formal coordination mechanisms to manage them, such as routines, preplanning, or standardization. Increasingly, however, collaborative work is characterized by high levels of interdependence, uncertainty, and time constraints, such as cooperative design [23] or emergency response [95]. In these dynamic contexts, interdependence of work is emerging and rapidly shifting over time [35]. Due to changes and evolution of collaborative plans, dependencies may emerge, sustain, or disappear as the activity advances.

The contingency of dependencies can be clearly evidenced in the motivating scenario, where patterns of dependencies among people are in fact quite volatile, either due to the changes of environment in which the work is done (environmental uncertainty), or the changes of the task that the contributors are trying to accomplish (task uncertainty) [62]. For instance, while the decontamination process at a given decontamination station is under way, a first responder reports that five new victims have been discovered and are on their way to the decontamination station. As a result, a request to deliver an additional operator is initiated in anticipation of exceeding its maximum capacity of victims the station can handle. Suddenly, the decontamination process becomes intertwined with the process of delivering the operator, and the five new victims cannot be decontaminated until the new operator arrives at the station. In this way, such unexpected events cause seemingly un-related tasks to suddenly become interdependent.

As a result, we believe that the contingency in the interdependence of work represents a major feature of complex geo-collaborative work, but existing coordination tools have not taken this into account seriously. When a collaborative activity involves an extended period of work with volatile dependencies, the potential dependencies (i.e. the collective set of all dependencies that could potentially happen during the whole activity) are a quite large number, but the actual dependencies (i.e. those dependencies that are live and need to be coordinated) are rather a relatively small number. Coordination tools could play a more effective role if they target coordination of actual dependencies, rather than potential dependencies. However, identifying the set of actual dependencies at any given moment is difficult due to the fact that such dependencies are unknown a priori and they emerge as a consequence of the evolving activity. It would be highly desirable for collaborative tools to be able to assess the characteristics of the activities,

computationally identify actual dependencies, and track their changes over time.

1.2 Research Objectives and Approach

By setting the scene in the previous section, the overall objective of this dissertation is to addresses the major challenges in coordinating complex, dynamic, and distributed geo-collaborative activities by developing an awareness-based coordination mechanism that can utilize the knowledge of activities and dependencies.

To achieve the research objective, this study follows the design science paradigm in information systems research [55]. Knowledge and understanding of the aforementioned coordination problems in geo-collaboration and their solutions are achieved through a set of design activities, including building, evaluation, and application of the designed artifact. Figure 1.2 provides the overview of the research approach in this study.

The research starts with justifying the relevance of the research to the problem domain, i.e. the awareness phenomena in collaborative activities. The unique characteristics of geo-collaboration motivate this study. Then the existing knowledge about theories, models, and methods of awareness is reviewed to provide the scientific foundation of this study. Based on the grounding work in the literature, the overall design framework is developed. Such a design framework help the author to identify knowledge gaps in existing studies, and develop concrete design issues that need to be addressed. These design issues motivates our computational framework.

Following the design-research paradigm, the expected contributions of this research are two-folded. On the one hand, the research activities in this study have the potential to extend our understanding of the awareness phenomena in complex collaboration. On the other hand, the proposed approach can be applied in many collaborative activities, such as emergency response, transportation management, to offer solutions to important real life problems.

The figure needs to be updated!

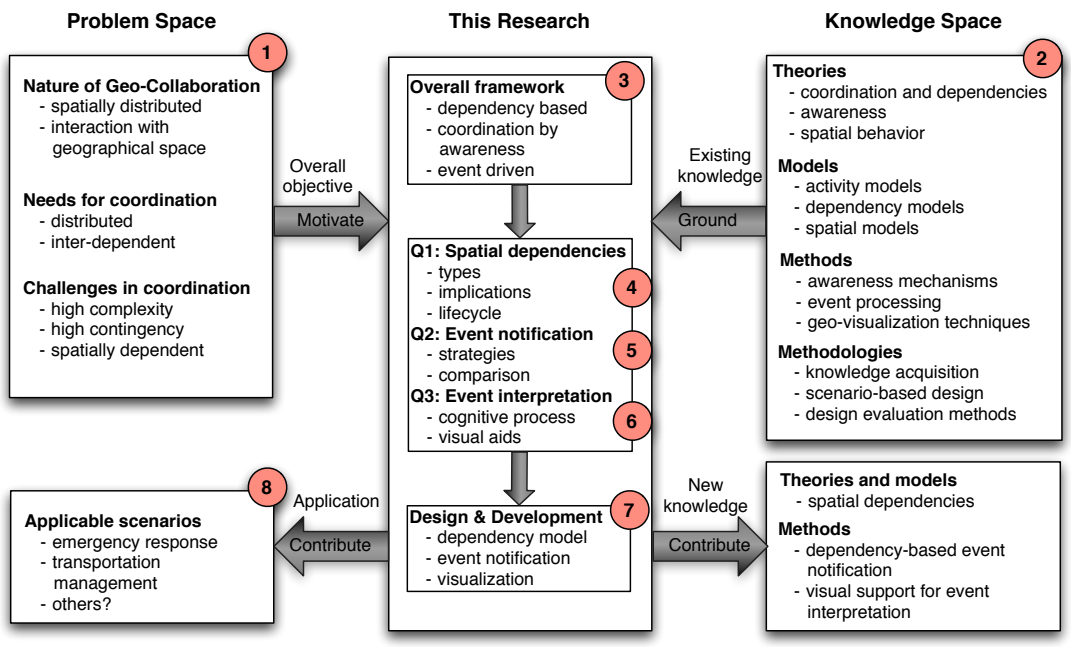


Figure 1.2. Overview of the research approach

1.3 Thesis Structure

The rest of this document is structured corresponding to how each chapter fits into the overall research framework. Chapter 2 presents the grounding work of this study, including the various studies on awareness phenomena. Chapter 3 provides the overall design framework and used it to review existing studies. Chapter 4-7 provide the details of our approach to awareness promotion. Chapter 8 demonstrates the use of the approach in several cast studies. Chapter 9 concludes the study.

Chapter 2

Understanding Awareness

2.1 Overview

Add a general diagram to layout the framework, discuss existing work in the framework, add a table to compare and summarize the literature

The concept of awareness has come to play a central role in both social and technical research in CSCW. However, what in CSCW labeled as 'awareness' has little in common, besides the fact that it represents some aspect of human interaction that is important for successful collaboration [93]. In a broad sense, two types of awareness can be distinguished in the CSCW research area: *social awareness* and *task-oriented awareness* [78, 93].

Social awareness addresses the availability of different kinds of information about the social context of the team members, e.g. awareness about what they are doing, if they are talking to someone, if they can be disturbed etc. *Social awareness* thus is conceived of as something that engenders "informal serendipitous interactions" [56] and "a shared space for community building" [29]. Awareness of the general social context is an important aspect of collaborative work, especially in domains where the the actors are engaged in cooperative work in a loose and broad sense, or domains where socialization is crucial [93].

However, when the tasks of collaborating actors become closely interdependent on each other, more urgent concerns need to be given to the aspect of *task-oriented awareness*. The *task-oriented awareness* focuses on practices through which actors seamlessly align and integrate their distributed and yet interdependent activities, e.g. awareness of things being done or in need of being done, of developments

within the joint effort that may be advantageous or detrimental for ones own work, of occurrence that makes ones work more urgent or leads to changes to the intended course of actions, etc. [93]. The major difference of *task-oriented awareness* from *social awareness* is that it focuses on activities performed to achieve a specific shared goal [20] and the actors being interdependent in their work [93], which lead to the unavoidable requirement for coordination.

In this study, we focuses on the **task-oriented aspect of awareness**, because the primary goal of supporting awareness in this study is motivated by the actors' being interdependent in their work and hence by the unavoidable requirements of coordinating and integrating their various actions in distributed geo-collaborative activities.

Within the scientific investigation into task-oriented awareness phenomena in collaboration, two lines of research can be identified. On one hand is the research aiming to establish conceptual understanding of the awareness phenomena from the cognitive and social aspects [89], and on the other hand is an increasing bearing on system design and development in particular technologies to promote awareness [83]. Although our work has an emphasis on the second aspect, i.e. the supportive technologies to promote awareness, we believe that a solid conceptualization of awareness phenomena in collaboration is extremely important for awareness promotion. System designers need to know what awareness might comprise and also how it is built and maintained in order to identify the specific awareness features they want to support [105].

As a result, before we move to the computational issues for awareness promotion, this chapter attempts to identify which of the existing theories and conceptualization of awareness in the literature is the most suitable for understanding the awareness phenomena in real world complex collaborative activities. A review and critique of what is currently known on the concept of awareness at both individual and team levels is presented, following which an integrated conceptual framework of awareness phenomena in complex collaborative activities is presented. In next chapter, we will show how such a conceptual framework helps us to evaluate existing computational awareness models and systems, and informs our design of the computational framework for awareness promotion.

2.2 Situation Awareness

Research into awareness at the individual level originated from the study of situation awareness (SA) in the human factors research community. Situation awareness is considered as knowledge created through interaction between a person and his/her environment, i.e. "knowing what is going on" in the situation [33]. A good general definition of situation awareness is as "the up-to-the minute cognizance required to operate or maintain a system" [1]. Although most of the situation awareness models in the literature are individual focused theories [89], it has also been well recognized as an important element in collaborative environments. For example, Gutwin and Greeenberg [49] view their workspace awareness as a specialization of situation awareness tied to the specific setting of the shared workspace. The concept of activity awareness proposed by Carroll et al. also subsumes situation awareness with an emphasis on aspects of the situation that have consequences for group work towards shared goals [20]. Hence, to understand the awareness phenomena in collaboration, it is important to start with understanding the practice of how individuals maintain and develop the situation awareness.

The human factors community has not settled on a common explanation of situation awareness, but we still can summarize some of the important characteristics that are well recognized in the literature, and also applicable to collaborative environments:

1. The products of awareness is the knowledge about the elements of the environment that is hierarchically structured [33].

2. The awareness phenomena should be seen as both product and process. As product, it is the knowledge that an actor can make use of. As process, it includes the cognitive processes through which the knowledge is achieved and developed) [1].

3. The process of achieving and developing awareness revolves around internally held, mental models, which contain activated information regarding current situations [102]. The activation of awareness information into the mental models are directed by the actor's activity [7].

We elaborate these three characteristics in the following of this section.

2.2.1 Hierarchically Structured Awareness Knowledge

Among the numerous attempts at specifying the products of situation awareness, i.e. what must be known to solve a class of problems posed when interacting with a dynamic environment [89], Endsley's three-level model [33] has undoubtedly received the most attention. The three-level model describes situation awareness as the operator's internal model of the state of the environment, comprising three hierarchical levels that is separate to the process used to achieve it [102]:

1. Level 1: *perception of relevant elements in the environment.* An actor must first be able gather perceptual information in the surrounding environment, and be able to selectively attend to those elements that are most relevant for the task at hand. At this stage, the information is merely perceived and no further processing takes place.

2. Level 2: *Comprehension of task-related elements in Level 1.* Level 2 involves the interpretation of the perceptual information from Level 1 in a way that allows an actor to comprehend or understand its relevance in relation to their tasks and goals.

3. Level 3: *Projection of the states in the near future.* Using a combination of Level 1 and Level 2 awareness-related knowledge and experience in the form of mental models, actors forecast likely future states in the situation.

Endsley's three-level model presents an intuitive description of situation awareness and has been applied in a plethora of different domains [116]. Its simplicity and the division of SA into three hierarchical levels allows the construct to be measured easily and effectively [31], and also supports the abstraction of situation awareness requirements and the development of design guidelines [89]. Furthermore, it has been extended in order to describe team situation awareness [32], and the three levels of awareness information are applicable in many collaborative situations [49].

Despite its popularity, the three-level mode has some important flaws. One of the key assumptions of the three-level model is the separation between depicting situation awareness as a product and the cognitive processes used to achieve it [89],

which leads to the inability to cope with the dynamic nature of situation awareness [102]. For example, Uhlarik and Comerford [114] suggest that the process of achieving situation awareness presented by the three-level model is both static and finite. Nevertheless, the model is also criticized by the ill-defined concept of mental models. Although Endsley's model emphasizes the critical roles of mental models in directing attention to critical elements in the environment (Level 1), integrating the elements to aid understanding of their meanings (Level 2), and generating possible future states (Level 3), the definition only includes the long-term knowledge that is formed by training and experiences, more important factors, such as the actor's goals, conceptual model of the current situation, are neglected [7].

2.2.2 The Development of Awareness

To address the dynamic nature of situation awareness, many researchers have used Niesser's perceptual cycle model [70] to clarify the cognitive components involved in the acquisition and development of situation awareness [102, 1, 49, 106]. According to the perceptual cycle model (Figure. 2.1), an actor's interaction with the world continues in an infinite cyclical nature. By perceiving the available information in the environment, the actor modifies its knowledge. Knowledge directs the agent's activity in the environment. That activity samples and perhaps anticipates or alters the environment, which in turn informs the agent. The informed, directed sampling and/or anticipation capture the essence of behavior characteristic of situation awareness.

Based upon Niessers perceptual cycle model, Smith and Hancock suggest that situation awareness is neither resident in the world nor in the person, but resides through the interaction of the person with the world [102]. Thus they viewing situation awareness as a generative process in 'an adaptive cycle of knowledge, action and information' [102]. In a similar fashion, Adams et al. [1] used a modified version of Niessers perceptual cycle model to describe how situation awareness works. They argue that the process of achieving and maintaining situation awareness revolves around internally held mental models, which facilitate the anticipation of situational events, directing an actor's attention to cues in the environment and directing their eventual course of action. An actor then conducts checks to confirm

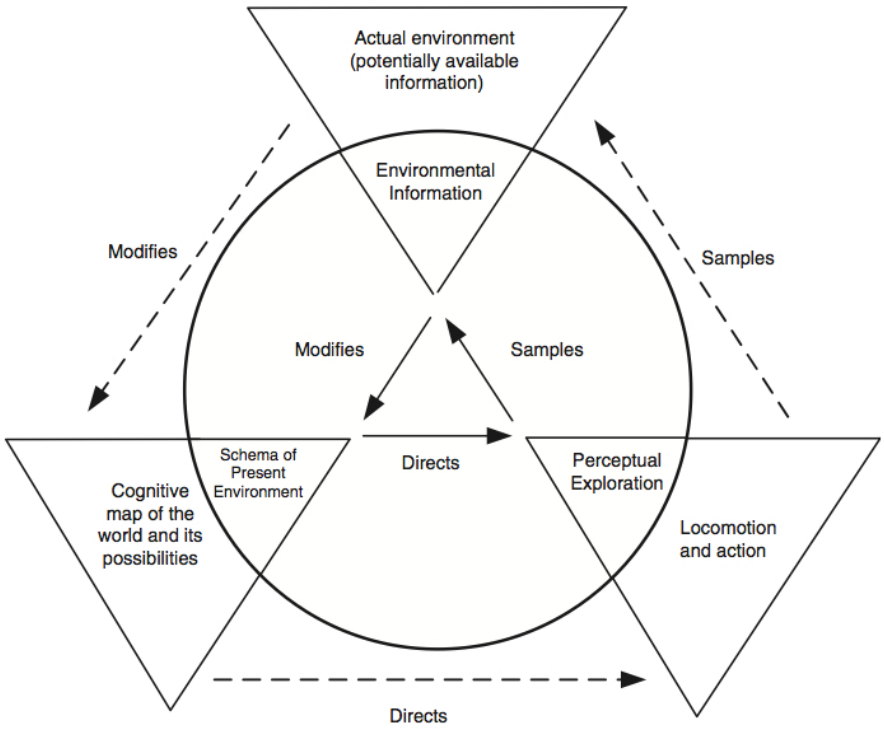


Figure 2.1. Niesser's Perceptual Cycle Model [89]

that the evolving situation conforms to their expectations. Any unexpected events serve to prompt further search and explanation, which in turn modifies the actor's existing model. Gutwin and Greeenberg used the perception-action cycle to explain how the awareness is maintained in a shared workspace, in which awareness knowledge both directs and is updated by perceptual exploration of the workspace environment [49].

Unlike the three-level model that depicts SA as a product separate from the processes used to achieve it, models based on the perceptual cycle view situation awareness as both process and product, offering an explanation of the cognitive activity involved in the development of situation awareness, and also a judgment as to what the product of SA comprises [89]. One of the key assumptions of these models is the interplay between the awareness information and the internal mental model of current situation. In the process of awareness development, some knowledge is activated and integrated into the mental model, while some becomes inactive or removed from the mental model. Although Smith and Hancock sug-

gested that the adaptation of awareness information into the mental model should be goal-directed, i.e. it must reside in the task environment rather than in the actor's head [102], little detail has been given about the cognitive processes that guide the selection and interpretation of awareness information into the mental models.

2.2.3 Activity Directed Awareness Process

Bedny and Meister propose a description of situation awareness based on the activity theory in an attempt to clarify the cognitive processes involved in the interpretation of awareness information [7]. Based on the activity theory, they purport that individuals possess goals that represent an ideal image or desired end state of activity, which direct them towards the end state or methods of activity (or actions) that permit the achievement of these goals. It is the difference between the goals and the current situation that motivates an individual to engage in the awareness process and take action towards achieving the goal. They conceptualize activity in three stages: the orientational stage, the executive stage, and the evaluative stage. The orientational stage involves the development of an internal representation or picture of the world or current situation. The executive stage involves proceeding towards a desired goal via decision-making and action execution. Finally, the evaluative stage involves assessing the situation via information feedback, which in turn influences the executive and orientational components.

Instead of considering the actor's internal mental model as a whole, they suggest that the mental model is comprised of several functional blocks as presented in Figure 2.2. Each functional block has a specific role to play in the development and maintenance of situation awareness and that the blocks orientate themselves towards the achievement of situation awareness. The interpretation of incoming information (function block 1) is influenced by an individuals goals (function block 2), conceptual model of the current situation (function block 8), and past experience (function block 7). This interpretation then modifies an individuals goals and experience and conceptual model of the current situation. Critical environmental features are then identified (function block 3) based upon their significance to the task goals and the individuals motivation towards the task goal (function block 4),

which directs their interaction with the world (function block 5). The extent to which the individual proceeds to engage the task goals is determined by their goals (function block 2) and their evaluation of the current situation (function block 6). The resultant experience derived from the individuals interaction with the world is stored as experience (function block 7), which in turn informs their conceptual model (function block 8).

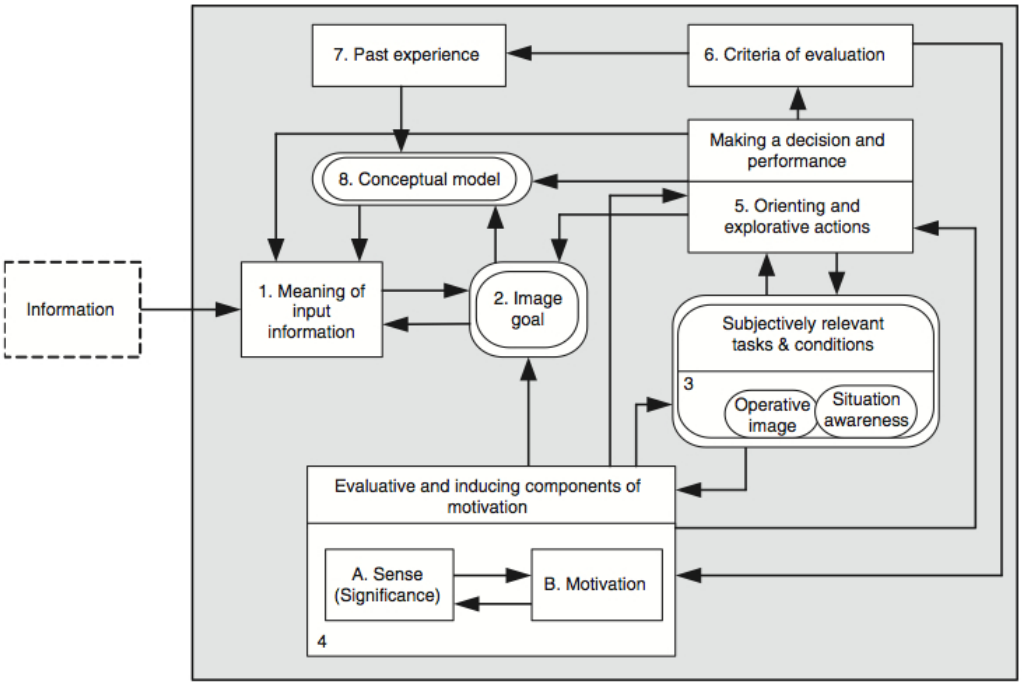


Figure 2.2. Bedny and Meister's Activity-Directed Awareness Process [7]

Based on the activity theory, Bedny and Meister clearly elucidate the functional blocks within the actor's mental model and their roles in the development of situation awareness [7]. Their model clearly shows how the actor's goals and activities play a central role to drive the process of awareness development. However, like most other situation awareness-related models, Bedny and Meister's activity theory model does not attempt to cater for, or explain the awareness phenomena in collaborative settings.

2.3 Awareness In Collaboration

The awareness phenomena in collaborative environments is indubitably more complex than situation awareness at the individual level. Salas et al. [87] point out that there is a lot more to team level awareness than merely combining individual team member's situation awareness. Beyond knowing what is going on in the environment, in their tasks, team members also need to develop an understanding of the activities of others, which provides a context of their own activities. This context is used to ensure that individual contributions are relevant to the group's shared goal as a whole [28]. This section reviews three prominent conceptualizations of awareness in collaboration that informs this study: team situation awareness, activity awareness, and distributed team awareness.

2.3.1 Team Situation Awareness

The research on team situation awareness (TSA) attempts to extend the theories and models of situation awareness to collaborative settings. Endsley et al. [32] suggest that, during team activities, situation awareness can overlap between team members, in that individuals need to perceive, comprehend and project awareness elements that are specifically related to their specific role in the team, but also elements that are required by themselves and by members of the team. Successful team performance therefore requires that individual team members have good situation awareness on their specific elements and also the same awareness for those elements that are shared. It is therefore argued that, at a simple level, team situation awareness comprises three separate but related components: individual team member's situation awareness; situation awareness of other team members; and situation awareness of the overall team (Figure 2.3).

Similar to individual situation awareness, team situation awareness should be considered as a dynamic process. Salas et al. [87] propose a framework of team situation awareness that comprises two critical processes, individual situation awareness and team processes. According to them, team situation awareness depends on communications at various levels. The perception of SA elements is influenced by the communication of mission objectives, individual tasks and roles, team capability and other team performance factors. The comprehension of awareness

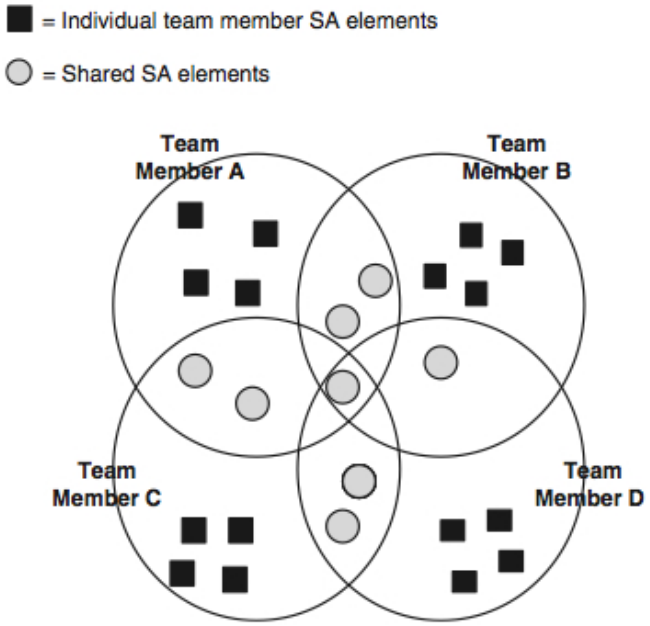


Figure 2.3. Team Situation Awareness (adapted from Endsley 1995 [33])

information (i.e. Level 2) is impacted by the interpretations made by other team members, so it is evident that SA leads to SA and also modifies SA, in that individual SA is developed and then shared with other team members, which then develops and modifies team member SA. Thus, a cyclical nature of developing individual SA, sharing SA with other team members and then modifying SA based on other team members SA is apparent.

Most attempts to understand team SA have centered on a 'shared understanding' of the same situation. Nofi [71], for example, defines team SA as: 'a shared awareness of a particular situation' and Perla et al. [76] suggest that 'when used in the sense of "shared awareness of a situation", shared SA implies that we all understand a given situation in the same way'. Shu and Furuta sugggested that TSA comprises both individual SA and mutual awareness and can be defined as 'two or more individuals share the common environment, up-to-moment understanding of situation of the environment, and another person's interaction with the cooperative task' [97].

Team situation awareness has been broadly recognized as a critical factor to understand awareness in collaborative environment, because its compatibility to

individual situation awareness, and the abstraction of individual awareness process from team processes [97, 89]. However, TSA also has many limitations.

1. First, a critical factor of team situation awareness is to define the configuration of shared situation awareness requirements, i.e. the degree to which team members understand which information is needed by which team member, or by the whole team. The identification of such shared awareness requirements is feasible in simple, small-scale collaborative scenarios However, for complex, real world collaborative scenarios, such a task becomes more intricate. In complex scenarios that involve numerous agents and artifacts working both collaboratively and dispersed geographically, viewing and assessing team situation awareness is actually too complex [89].

2. Second, the role of team processes, such as communication, mutual monitoring, in maintenance and development of team situation awareness, is only barely touched. It is recognized that an increased level of team processes will lead to enhanced levels of team situation awareness. However, the specific relationships between team situation awareness and team processes remains largely unexplained [89].

3. Last, the team situation awareness adopts the knowledge-in-common view of shared mental models [68], i.e. it focuses on how the shared understanding of the same situation is developed. However, as argued by Mohammed and Dumville, the knowledge-in-common view may be appropriate for only certain task domains and types of groups [68]. For example, in teams with high level of division of work, the distribution of knowledge and skills across the team typically is not uniform, as a result, a high level of overlapping knowledge in such teams might be inefficient.

2.3.2 Activity Awareness

To address the problem of team situation awareness that posits 'knowledge in common' as a basis for awareness, Carroll et al. proposes a new framework for understanding awareness in collaborative environment, based on the concept of 'activity awareness' [20, 21]. The major distinction between team situation awareness and

activity awareness is that, in realistically complex circumstances, instead of merely sharing relatively static and stable constructs such as knowledge in common, people share their activities [21]. In framing activity awareness, they appropriate the concept of *activity* from Activity Theory to emphasize that collaborators need to be aware of a whole, shared activity as complex, socially embedded endeavor, organized in dynamic hierarchies, and not merely aware of the synchronous and easily noticeable aspects of the activity [22].

Similar to the activity-directed SA model proposed by Bedny and Meister [7], activity awareness emphasizes the importance of using the concept of activity to structure the products and processes of awareness phenomena. The ultimate motivation of human actors to acquire and maintain awareness in the collaborative environment is to achieve their shared goals by performing their activities. As a result, the context surrounding a collaborative activity (e.g. the manner in which a shared activity is decomposed into smaller inter-related tasks, how these subtasks are assigned or adopted by collaborators, and when and how distributed subtasks are interdependent on each other), becomes the most important aspects of the situation that the team members need to be aware of [20].

By shifting the focus from shared knowledge to shared activity, activity awareness aligns the development of awareness with the development of collaborative activities. Most basically, activity awareness is achieved and developed through the join construction of common ground - shared knowledge and beliefs, mutually identified and agreed upon by members through a rich variety of communication protocols [21]. In long-term, open-ended activities over significant spans of time, the construction of shared practices, social capital, and human development become also important to enhance team member's activity awareness.

Activity awareness with its basis on Activity Theory, primarily focuses on the social aspect of the awareness phenomena, i.e. how the team members' awareness of the social context of collaborative activities is developed through common grounding, construction of shared practices, social capital, and human development. Although the authors claim that activity awareness subsumes situation awareness [20], little discussion has been given to how these higher-level social processes are connected to the cognitive processes to maintain situation awareness. Issues, such as how the state change in the external environment can lead to

a team member's internal goal change, which could furthers lead to re-planning of their activities, cannot be explained.

Furthermore, the activity awareness focuses on the sharing of activities, i.e. the importance of a common picture of the shared collaborative activities. However, such a common picture is usually distributed in the whole group, instead of in any single actor's mind [106]. Each actor in the group has their own awareness, related to the goals they are working towards. However, this seldom includes the whole picture of the collaborative activity, and only when all the actors' awareness knowledge is meshed up together, the common picture emerges. Activity awareness framework provides little support to explain how the activity knowledge is distributed across multiple actors.

2.3.3 Distributed Team Awareness

A more recent theme to conceptualize awareness in collaboration is the concept of distributed or systemic team awareness [106, 6]. Distributed team awareness approaches are borne out of distributed cognition theory [58], which describes the notion of joint cognitive systems comprising the people in the system and the artifacts that they use. Within such systems, cognition is achieved through coordination between the system units [6] and is therefore viewed as an emergent property (i.e. relationship between systemic elements) of the system rather than an individual endeavor. Distributed team awareness approaches therefore view awareness in collaboration not as a shared understanding of the situation, but rather as a characteristic of the socio-technical system itself [6]. Whilst recognizing that team members possess their individual SA for a particular situation and that they may share their understanding of the situation, distributed team awareness assume that awareness is distributed across the different human and technological agents involved in collaborative systems [106].

The main difference between distributed team awareness and other TSA and activity awareness models relates to the concepts of *compatible* and *shared* awareness. *Shared* awareness accounts suggest that efficient team performance is dependent upon team members having the same awareness knowledge. Distributed team awareness, on the other hand, postulates that, within collaborative systems, differ-

ent team members have unique, but *compatible* awareness, regardless of whether the information that they have access to is the same or different [106]. Team members experience a situation in different ways, as defined by their own personal experience, goals, roles, tasks, training, skills, and so on. So whilst some of the information required by two different team members may be 'shared' in the sense that they both need to attend to it as part of their job, their resultant understanding and use of it is different [90]. Ultimately, the picture developed by each team member is unique for themselves. *Compatible* awareness is therefore the phenomenon that holds distributed systems together. Each team member has their own awareness, related to the goals that they are working towards. Although different team members may have access to the same information, differences in goals, roles, the tasks being performed make them view it differently. In this way, each team members awareness is different in content, but is compatible in that it is all collectively required for the system to perform collaborative activities successfully.

While the distributed team awareness emphasizes the distribution of awareness, it does not discount the social interactions among different team members. The term 'transactive' awareness is used to describe the notion that distributed team awareness is acquired and maintained through *transactions* that arise from communications or other team processes [90]. A transaction in this case represents an exchange of awareness between one agent and another (where agent refers to humans and artifacts). Agents receive information, it is integrated with other information and acted on and then passed on to other agents. The interpretation on that information changes per team member. The exchange of information between team members leads to transactions in the SA being passed around. For example, an agent may perceive certain awareness element in the environment, interpret the meaning, and then pass it to another agent via a transaction. The second agent then builds its own interpretation upon the first agent's interpretation, and may start a new transaction to pass the awareness to other agents. Hence, it is the systemic transformation of awareness elements as they cross the system boundary from one team member to another that bestows upon awareness in collaboration an emergent behavior [106].

The concept of distributed team awareness has been investigated by the authors in a number of domains, including naval warfare [105], energy distribution [88], and

air traffic control [106]. The major strength of the approach is related to the systemic approach that it advocates, which is more suitable to analyze the awareness phenomena in complex, real world collaborative activities [106]. However, the main weakness, as admitted by the authors, is also related to it complexity [90]. Similar to other team situation awareness models, it uses concepts as the basic unit to analyze awareness elements, which often leads to extremely large networks in order to represent all the concepts and their relationships. A possible remedy is to integrate the distributed team awareness with the activity-directed models and switch the focus from concepts to activities to understand the awareness phenomena.

2.4 An Integrated Conceptual Model of Awareness

By reviewing the existing theories and conceptualizations of awareness, we believe that, instead of adopting one particular viewpoint, a suitable conceptual model for understanding the awareness phenomena in real world complex collaborative activities requires the integration of multiple constructs in the literature. Specifically, we identify the following requirements:

1. *The integration of individual cognitive processes and social processes.* As most theories and models of awareness in collaboration claim that individual situation awareness is still an important component in collaborative environment, the conceptual model should be able to account for both cognitive processes at the individual level and social processes at the team level, and emphasize on how these two aspects interplay with each other.

2. *The integration of compatible and transactive aspects of awareness phenomena.* We agree with the distributed team awareness approaches [90] on that, because of the differences in goals, roles, the tasks being performed make, each team members awareness is different in content, but at the same time is compatible for the team to perform collaborative activities successfully. Hence, the conceptual model should be able to account for how the awareness is distributed across multiple team members, and meanwhile can interact with each other via transactions.

3. *The integration of awareness and activity.* We resonate with the activity-directed SA model [7] and the activity awareness framework [20] to emphasizes the importance of using the concept of activity to structure the products and processes of awareness phenomena. As the purpose of human actors to acquire and maintain awareness in the collaborative environment is to achieve their shared goals by performing their activities, it is very natural to use the activities to structure their awareness requirements.

To satisfy these requirements, we propose an integrated conceptual model of awareness in complex collaborative activities, by combining multiple constructs in the literature. In general, the model has two major components: (1) an integrative model of activities, local scopes of work, and dependencies that form **the field of collaborative work**, (2) and **a set of awareness processes** built on top of it. The goal of the former is to establish the necessary knowledge that is needed to understand the content of awareness, i.e. *aware of what*, and the later focuses on the awareness processes, i.e. *how awareness is achieved and developed*. In the following of this section, we elaborate each component in more details.

2.4.1 The Field of Work

Following the activity-directed SA model [7] and the activity awareness framework [20], we focus on the concept of activity to structure the content of awareness, i.e. the field of collaboration work, built on top of three interrelated concepts (Figure 2.4):

1. We consider *activity* as the basic unit of analysis to support coordination in geo-collaboration.

2. Due to the distributed nature of geo-collaboration, the various activities have different implications on different actors, and form their respective *local scopes of work*.

3. Activities in different local scopes of work are interdependent due to various types of *dependencies* existing among them. Dependencies serve as the bridges between different actors' activities and make their local scopes of work overlapping with each other.

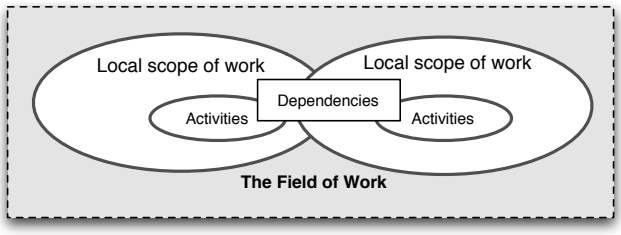


Figure 2.4. The Field of Work

2.4.1.1 Activity as the basic unit

In our approach, we subscribe to Activity Theory to conceptualize human activities [69]. From this perspective, the basic structure of an activity can be defined as several basic elements and mutual relationships between them.

1. **Actions** An action specifies a particular way of doing something. An action can be either basic or complex. A basic action can be directly executed by one or multiple actors. A complex action needs to be decomposed into subsidiary actions. When an action is specified as a sub-component of a higher action, this restricts the higher action to that particular course of doing. An action is assigned to a list of actors who are responsible to executing it.

2. **Actors** are defined as entities capable of performing actions and capable of making decisions on the performance of their actions.

3. **Resources** are anything that can be used in the transformation process of an action, including both material resource and resources for thinking. For example, in the case of response emergency, resources may include rescue vehicles, medical equipments, and information such as the locations of victims.

2.4.1.2 Local scope of work

Although various activities can be identified in a complex collaborative environment, each actor usually only engages in a small set of them. In fact, one of the fundamental motivations for team work is to decouple a complex problem into a set of smaller ones that are much easier to manage and tackle. One result of the

decoupling is that the work of actors is distributed and each actor is only interested in the states of a small set of activities, resources, and conditions that are relevant to their roles, current goals and tasks. To characterize the distributed nature of team awareness, we define the local scope of work for each actor as the set of activities, resources, and conditions that have direct or potential impact on the actors work.

The concept of local scope of work have several characteristics of the distributed nature of emergency response activities:

1. First, the local scope of work is a relative concept that changes with the actors current goals. Whenever an actors goal is changed, the local scope of work may also be changed.

2. Second, the local scope of work may cover multiple portions of the collaborative activity as a single actor may focus on several activities at the same time.

3. Third, the local scopes of two actors can overlap with each other. It is common that the same activity, resource, or condition falls into the local scopes of different actors, even though it may have different impacts on these actors. Actually, these overlapping elements across multiple local scopes of work play an important role in supporting the transactive awareness process.

2.4.1.3 Dependencies

Although activities are largely distributed and belong to local scopes of different actors, they cannot be performed without interacting with each other. Although an actor is only responsible for or interested in the activities within her/his local scope of work, the activities outside the local scope can still potentially impact her/his work through dependencies.

Dependencies have been studied by many researchers with aspect to coordination, from organizational studies ([118, 26]), multi-agent systems (MAS) ([**?**, 99, 98]), and computer-supported cooperative work (CSCW) ([80, 24, **?**]). Dependencies within a collaborative process can be of various forms. In general, we

can summarize three types of dependency relationships that have been well recognized in the literature: temporal relationships, resource-related relationships, and goal-related relationships.

1. *Temporal dependencies.* The activities of the actors might be interdependent due to the constraint of ordering them in a certain order [99]. In some case, certain activities cannot be started until others are finished (the problem of sequencing) or a certain group of activities have to be performed at the same time (the problem of synchronization).

2. *Resource dependencies.* Resource dependencies can be analyzed in terms of common objects, i.e. resources, that are used by multiple actors or involved in multiple actions. An example is when two or more users simultaneously want to alter the same part of a document in a collaborative authoring system [**?**]. These common objects constrain how each activity is performed. Different patterns of use of the common objects by the activities will result in different kinds of dependency relationships [**?**]: A *fit dependency* occurs when multiple activities collectively produce the same resource. A *flow dependency* arises whenever one activity produces a resource that is used by another activity. A *sharing dependency* arises whenever multiple activities all use the same resource.

3. *Goal dependencies.* Goal dependencies involves the subsidiary goals that might be interdependent across various actors or their tasks [99]. A goal-related dependency reflects the fact that the depender depends on the dependee to bring about a certain state in the world. Unlike the temporal or resource-related dependencies in which the depender knows what tasks are involved in these relationships, the depender in a goal-related dependency does not care how the dependee goes about achieving the goal, i.e. the dependee is free to, and is expected to, make whatever decisions are necessary to achieve the goal [118]. A goal-related dependency is usually characterized by the structural relationship resulting from goal decomposition of the whole shared goal, i.e. task A depends on the other task B because B is a means to achieve a subsidiary goal that must be satisfied in order to perform A.

2.4.2 Characteristics of the field of work

1. Partiality of local scopes One of the central concepts in our approach is the local scopes of different actors in a collaborative activity. Comparing with the filed of work representing the overall collaborative activity, the local scope of an actor represents the part of the field of work that the actor is participated in.

the local scopes are partial representations of the whole field of work. This is the foundation of group work. If no difference, then why do we need to cooperate?

2. Connectivity of local scopes

One of the basic assumptions behind our event propagation mechanism is that the different actors' local scopes in a collaborative activity are compatible with each other.

1). Overlaps, boundary objects

Although the different actors are interested in and capable of performing different actions in the collaborative activity, there are always some actions sitting in between different actors' local scopes as boundary objects. It is these boundary objects that provide the bridges for event propagation. As an event interpreted by one actor is associated with a boundary object, it becomes relevant to the other actor whose local scope also covers this boundary object.

2). Dependencies The phenomena of overlapping local scopes can be explained by the dependencies existing between different actors' actions. When the action act_1 of an actor(ar_1) depends on another actor ar_2's action act_2, ar_1 will also be interested in knowing the information related to act_2, such as the fact that act_2 has been completed or it has problem to be performed. In this way, the act_2 can be considered as a boundary object that overlapping the two actors' local scopes.

3. Dynamics of the field of work

Figure 2.5 illustrates how the different activities are distributed into different local scopes of work (represented as dotted circles) and how they are connected by the various dependency relationships in the motivating scenario. As we can see from the example, the different actors in the scenario, the victim manager, the decontamination manager, and the transportation manager, have very different roles to play, and therefore their local scopes vary significantly. However, due to the dependencies among their activities (For example, the decontamination manager relies on the transportation manager to deliver the victim to the decontamination

station; the victim manager relies on the decontamination manager to perform decontamination on the victim), their local scopes are overlapped with each other.

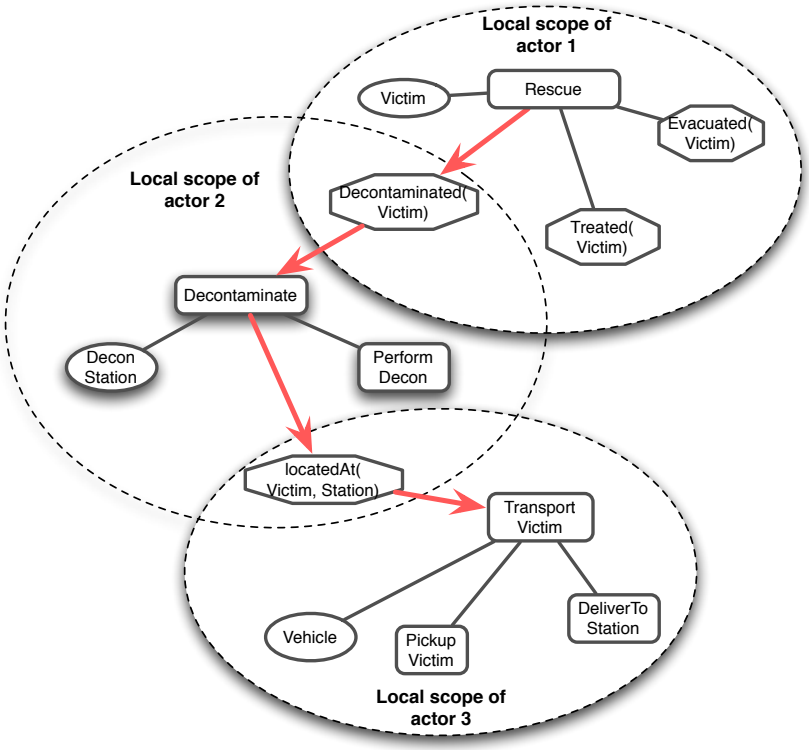


Figure 2.5. An Example: The Field of Work

2.4.3 Awareness Processes

Built upon the field of work, a set of awareness processes can be identified to describe how the awareness is acquired and developed in a cyclical way (Figure 2.10). In general, we can identify the awareness development cycles at both the individual and team levels.

2.4.3.1 The Development of Individual Awareness

At the individual level, each team member develops his/her own awareness in the similar way as described in the Neisser's perceptual cycle model [70]: the development of awareness starts with the perception of selective elements in the

actor's local scope of work, which is then interpreted with the help of the actor's existing knowledge. The result of interpretation then help the actor to make certain decisions and perform actions, which then further update the actor's local scope of work (Figure 2.6). An important difference in our model from the original perceptual cycle model is that, instead of action directly stimulated by the acquired awareness knowledge, we emphasize the individual's planning process before the action performance, in which the individual needs to make decisions on what to do based on the interpretation of awareness information, such as whether he/she needs to change the goal, perform re-planning, or ask for help etc.

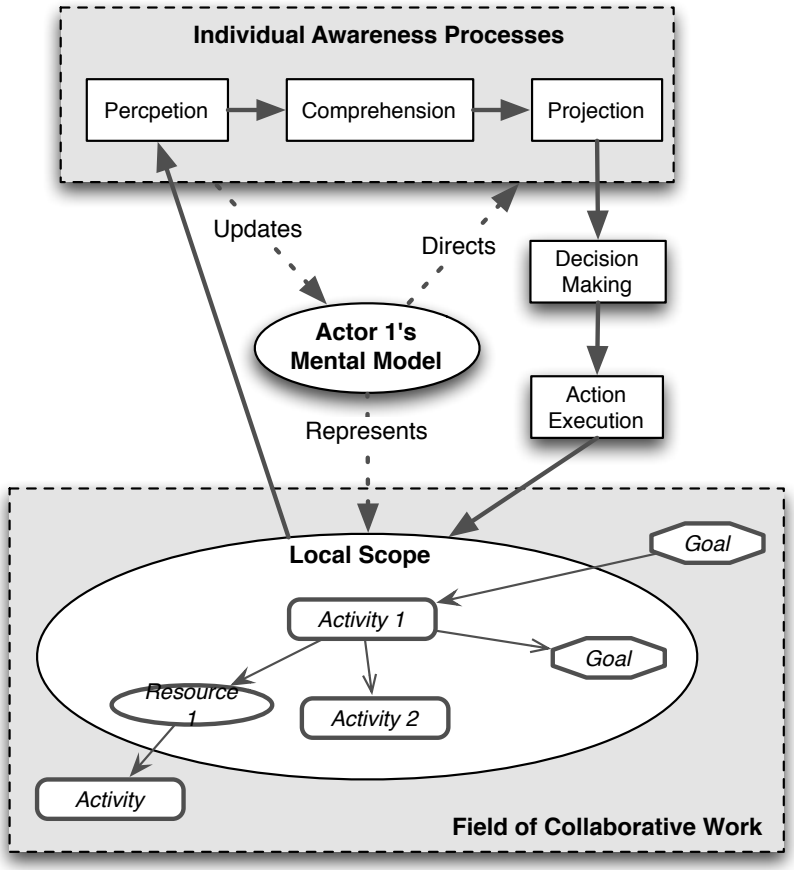


Figure 2.6. The Development of Individual Awareness

The individual processes need to be modified.

1. *Perception.* The process of perception is very similar to the concept in the

perceptual cycle model. An actor must first be able gather perceptual information in the surrounding situation, and be able to selectively attend to those elements that are most relevant for the task at hand. The only difference is that the perception is constrained by the actor's local scope, i.e. he/she can only perceive the information about the environment, acitivities, or other objects defined in his/her local scope.

2. *Comprehension.* Comprehension refers to the mental activities where perceived awareness information is processed for the purpose of understanding its meaning in the context of the individual's goals and activities.

3. *Projection.* The process of predicting likely future states in the situation.

4. *Decision Making.* The process of decision-making involves two major tasks. The first is to elaborate the actor's plan based on the interpretation. This involves decisions such as commitment to certain activity, focus switching, re-planning. The second task includes decisions about whether the actor wants to propagate the interpretation to other actors, which is important for initiating the awareness development cycle at the team level.

5. *Action.* Based on the result of decision-making process, the actor may perform some action to manipulate his/her local scope, which can lead to further changes that will start another round of perception.

2.4.3.2 Team Awareness Propagation

Beyond the individual processes of awareness development, our conceptual model allows the awareness propagation across multiple actors in a team. In general, we can identify three basic trajectories of how the propagation can work: (1) via explicit communication, (2) via implicit feed-through via action execution, and (3) via the manifestation on boundary objects in overlapping local scopes.

Elaborate on how each approach in more detail

Based on the interpretation of awareness information, an actor may decide to propagate the change to other actors by indicating how the other actor's activities can be impacted by his/her interpretation. The result of propagation is then

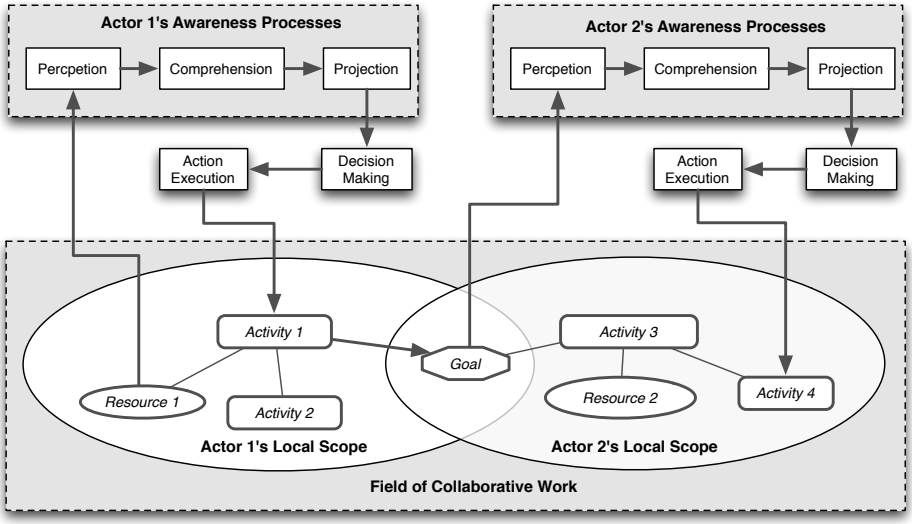


Figure 2.7. Team Awareness Propagation via Feed-through

perceivable by the other actor and may initiate the other actor's individual awareness development cycle. As the other actor develops his/her individual awareness, he/she may decide to propagate his/her interpretation back to the actor or other actors in the team, or he/she may perform actions that lead to changes in other actors' local scopes. In either way, new individual cycle may be initiated and the awareness development continues at the team level.

The development of awareness at the team level is guided by the dependencies among activities and the shared activities in overlapping local scopes. The dependencies allow the actors to cascade the awareness interpretation across multiple activities. The shared activities in overlapping local scopes enable the propagation process to cross the boundary of multiple local scopes.

2.4.4 Discussion

Figure 2.10 shows the integrated conceptual framework by combining the aforementioned components together.

Figure 2.11 demonstrates how the awareness processes work on top of the three constructs. In the beginning, Actor 1 perceives some unexpected event happening in the environment, and attempts to interpret it as whether and how it can impact the activities within her local scope of work. After understanding the event, she

This example needs to be revised and elaborated!

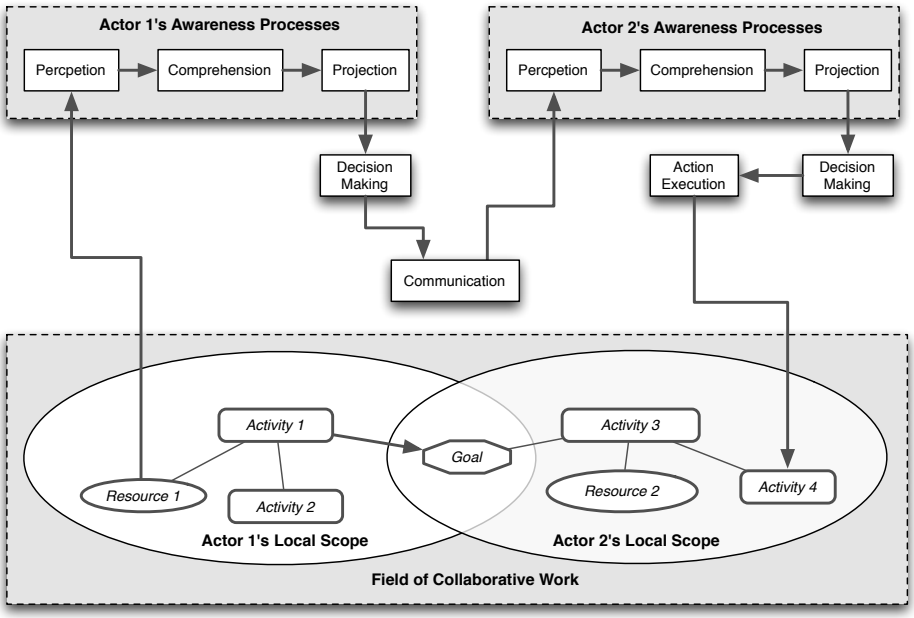


Figure 2.8. Team Awareness Propagation via Communication

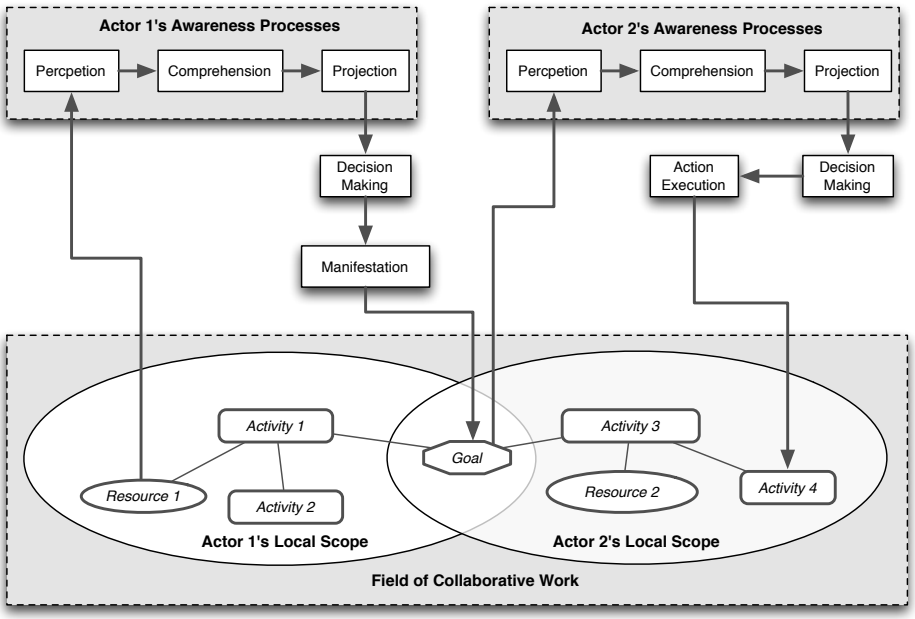


Figure 2.9. Team Awareness Propagation via manifestation

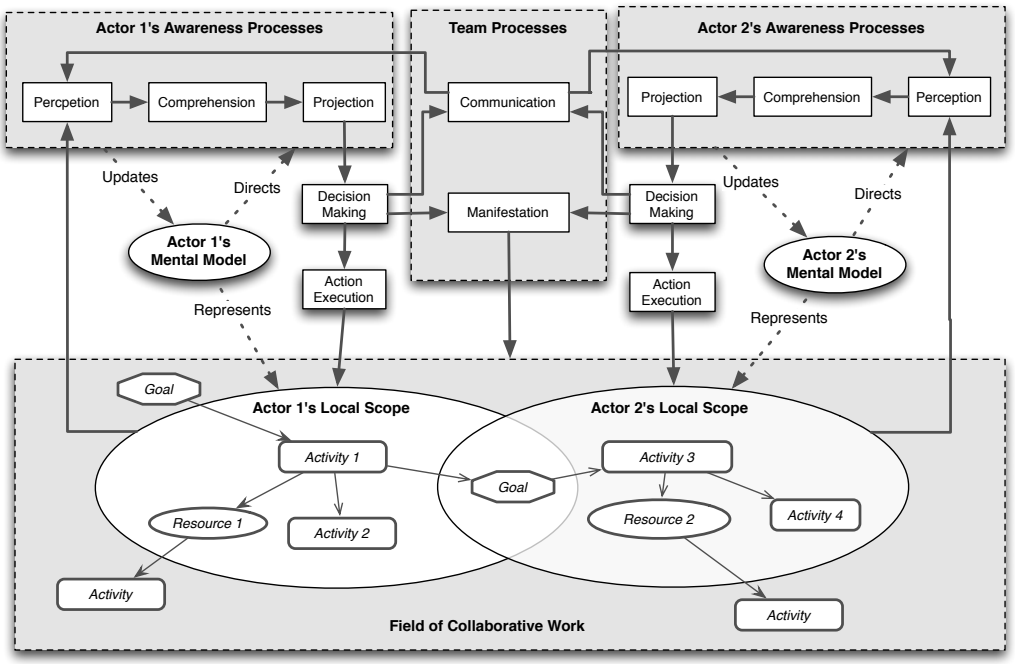


Figure 2.10. The Integrated Conceptual Model of Awareness

associates it with a state change of Activity 1. Furthermore, she predicts how the state change of this activity will impact other activities because of the dependencies among them. After Actor 1s interpretation, the event is likely to impact another activity (Activity 2), and Actor 1 decides to propagate it. The propagated event falls into Actor 2s local scope of work. Upon receiving this interpretation from Actor 1, Actor 2 first needs to understand where this event comes from by backtracking the interpretation, and evaluate how this event will impact his own line of work, which leads to a new projected state change of Activity 3. The similar process then is propagated to Actor 3s local scope of work. In this way, the teams awareness of the initial external event is collaborative developed as the relevant actors gradually attach their interpretations to it.

The proposed conceptual framework shows the capabilities to satisfy the three requirements we presented in the beginning of this section.

1. First, the conceptual model provides two ways to connect the individual cognitive processes and team processes. On one hand, the connection can be implicit as team members interact with the filed of work. The development of

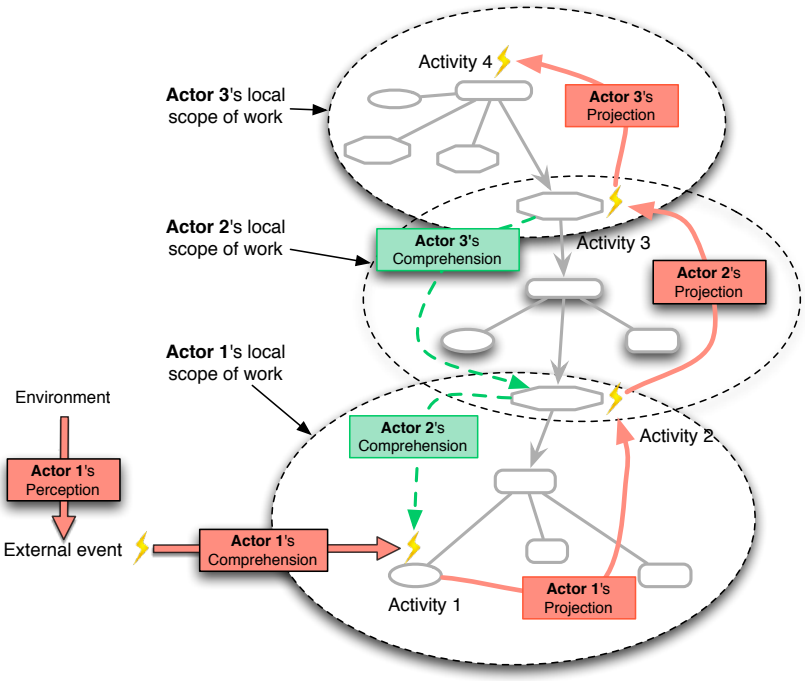


Figure 2.11. An Example: The Awareness Processes

an actor's individual awareness can manipulate the field of work, which can then be perceived by other actors in their separate individual awareness processes. In addition, the actors can explicitly propagate their interpretations to other actors within their individual awareness processes.

2. Second, the concept of local scope allows us to define how the awareness knowledge is distributed across multiple actors. The different actors often attend to different sets of activities as their local scopes. At the same time, the actors' local scopes are often overlapped due to the various dependency relationships that may occur between activities, this provides the mechanism to allow awareness transactions among multiple actors. In this way, the compatible and transactive aspects of awareness phenomena are integrated.

3. Last, our structure of the field of work is based on the activity theory, which enables the integration of awareness and activity.

Chapter 3

Designing for Awareness Support

Following the conceptual model of awareness phenomena in complex collaborative activities, we describe the design space of awareness support in this chapter. The goal of this chapter is to identify key design issues to support awareness in distributed, complex collaborative activities, by mapping them to the major components in our conceptual model presented in Chapter 2. We first present a framework of the whole design space with the major design issues identified. Then we discuss how these design issues have been addressed (or partially addressed) in existing awareness support systems, and the design challenges that motivate our approach.

3.1 The Design Framework

By conceptualizing the awareness phenomena in distributed, complex collaboration as a continuous development of awareness knowledge (i.e. the knowledge about the environment, activities, and their dependencies) through a variety of cognitive and social processes, the design issues for awareness support can be organized at two levels: the *individual* level and the *team* level. The former focuses on supporting the cognitive processes of individual team members to develop their own awareness, while the latter provides support for the social processes in which team members interact with each other to achieve awareness at the team level.

3.1.1 Designing for individual processes

At the individual level, the role of computer support can be understood in term of designing 'cognitive artifact' [72], which is defined as an artificial device designed to maintain, display, or operate upon information in order to aid cognition. Norman argues that an important design consideration for cognitive artifacts is the human action cycle that emphasizes the two sides of human action [72]. One side is the 'evaluation' side of action by perceiving, interpreting and evaluating the state of the environment. The other is the 'execution' side that of acting upon the environment. Norman's human action cycle matches well with the major steps of awareness development at the individual level, where the 'evaluation' side refers to the *perception*, *comprehension*, and *projection*, and the 'execution' side includes the *decision making* and *action execution*.

Perception

The achievement of individual awareness starts with the ability of individuals to perceive key features in the environment. The primary goal of awareness support in the perception process is to ensure that the awareness information is perceivable to the users. This goal can be achieved from two aspects: *selection* and *presentation* [12]. The former focuses on filtering out the information so that only the relevant information is perceived by the users, while the latter involves strengthening the stimulus to ensure that information is perceivable.

1. Support for *selection* is based on the assumption that perception is a selective process that depends on the requirements of the current working situation and the tasks at hand [33]. Not all information is of relevance to a user, hence the pool of all awareness information has to be processed to filter the relevant information [12]. Support for selection is to delegate the effort of filtering out irrelevant information to computer systems, so that human actors can focus on processing only the relevant set of information to avoid information overload.

2. Support for *presentation* is to strengthen the stimulus in the user interface to ensure important awareness information is perceivable. Existing studies

in cognition have shown that the salience of elements in the environment will have a large impact on which portions of the environment are initially attended to, and these elements will form the basis for perception [52]. As a result, the way in which information is presented via the interface will largely influence the perception process by determining which part of the environment will draw the user's attention.

Comprehension

Comprehension is to understand the meaning of perceived awareness information within the context of a user's current goals and activities [73]. In the comprehension process, new information must be combined with existing knowledge to develop a composite picture of the situation [33]. Hence, the computer support for comprehension is to help the users establish the connection between new awareness information and existing knowledge that forms the inferential framework [20]. In general, there are several ways that the comprehension can be supported by computer systems:

1. *Representation.* First, the computer system can support human comprehension by providing external representations of the existing knowledge. Instead of merely relying on the internal representations of human users, the external representation can serve as the information store, so that the internal representation at a given time can be quite sparse, perhaps containing only detailed information about their current focus [52], or pointers to locations of other important information in the external representation [63]. In this way, the limited working memory resources of human users are freed up for other aspects of cognition [63].

2. *Linking.* Second, the computer system can directly aid the linking between new awareness information and existing knowledge by presenting the awareness information along with the contextual information that is potentially relevant to understand its meaning [110]. Supporting comprehension by linking has its basis on the design principle of offloading cognitive processes onto perceptual processes [63]. By explicitly linking the awareness information

to contextual information for interpretation, some complex cognitive processes, such as searching for and activating the relevant portion of existing knowledge, can be replaced by simple pattern recognition processes [52].

Projection

Projection is the process that the individual predicts the future states of other activities based on the comprehension of awareness information. As argued by Endsley [33], this is the most difficult and taxing parts of situation awareness because it requires a fairly well developed mental model of the activities and relationships among them, and the capabilities to perform reasoning. System-generated support for projecting future events and states of the system can then target on supporting the development of the mental model by representing the activities and relationships, or supporting the reasoning processes.

1. *Representation.* Similar to the comprehension process, the projection process can be supported by providing external representations of the existing knowledge to enhance human cognition. However, unlike the representational support for comprehension that focuses on individual activity elements, the knowledge representation for projection emphasizes the various relationships and dependencies among the activity elements, so that the users can infer how the state on one activity can lead to possible changes on other activities' states.

2. *Reasoning.* The analytical reasoning is central to the projection process, through which users identify possible alternative future scenarios and the signs that one or another of these scenarios is coming to pass [109]. As a result, one of the critical requirements for supporting projection is to provide the analytics tools and techniques that allow the users to synthesize information and derive insight from it.

3.1.2 Designing for team processes

At the team level, the roles of computer in awareness support can be characterized by supporting the three basic types of team processes to propagate awareness

information as described in Section 2.4.3.2: *feedthrough*, *communication*, and *manifestation*.

Feedthrough

The process of feedthrough is to make consequences of individual activities apparent to other participants [28]. In co-located collaborative environments, this usually can be achieved without computer intervention, as collaborators can readily see each other's activities and artifacts they are working on [93]. However, in distributed collaboration, computer support becomes inevitable to enable the process of feedthrough. The role of computer to support feedthrough is to *broadcast* the effects of individual actions and make them visible to each other.

Communication

Communication is the prevalent form to propagate awareness information, in which people explicitly talk about awareness elements with their collaborators [49]. Computer mediated communication has been an important component in almost every distributed collaborative system, by providing team members a *medium* to communicate with each other remotely.

Although we consider communication as an important process for the team members to propagate awareness information, it is usually supported separately from the awareness systems as a standalone feature in collaborative applications. As a result, we do not put much emphasis on supporting communication in the following discussion.

Manifestation

Manifestation refers to a more subtle means to propagate awareness information among team members. Instead of directly performing actions to impact each other via feedthrough, or explicitly communicating with each other, the team members can make some aspects of their individual awareness visible to others, so that anyone who is interested in these aspects, or who is monitoring the field, can perceive the information. The aspects of individual awareness that can be made visible in the manifestation process can be the raw awareness information perceived by

a team member, or his/her interpretation of the awareness information during comprehension/projection, or the results of decision-making. To support the manifestation process, the computer system needs to provide the following functions:

1. *Externalization*. The computer system needs to provide the tools to allow the users to externalize aspects of their individual awareness and make them visible to other users.

2. *Visibility*. The computer system should allow the users to control the visibility of their manifested information. They can specify who can see what piece of the information they make visible.

Table 3.1 summarizes the whole design space with the major design issues for each element identified.

Awareness Process	**Supporting Aspects**
Perception	*1. Selection*: filtering out the information so that only the relevant information is perceived by the users *2. Presentation*: strengthen the stimulus in the user interface to ensure awareness information is perceivable
Comprehension	*1. Representation*: providing external representations of the existing knowledge to support human comprehension *2. Linking*: presenting the awareness information along with the contextual information that is potentially relevant to understand its meaning
Projection	*1. Representation*: providing external representations of activities and their relationships *2. Reasoning*: provide analytics tools and techniques to help users perform reasoning
Feedthrough	*1. Broadcasting*: making consequences of individual activities apparent to other participants
Communication	*Medium*: providing team members a medium to communicate with each other remotely
Manifestation	*1. Externalization*: providing tools for the users to externalize aspects of their individual awareness and make them visible to other users *2. Visibility*: controlling the visibility of manifested information, i.e. who can see what piece of the information

Table 3.1: Design Space for Awareness Support

3.2 The State of Art

By outlining the design space for awareness support in Section 3.1, we can use it as a framework to review existing studies and awareness systems by looking into how the design aspects in various cognitive and social processes have (or have not) been supported. Before that, we need to make a distinction between two types of awareness models for organizing and processing awareness information: *space-based* and *event-based* models, because it is the underlying awareness model that determines how the awareness processes are supported in an awareness system [45].

3.2.1 Awareness models

Most awareness systems rely on certain computational models for organizing and process awareness information. An awareness model usually provides a representation of the field of work, and specifies how awareness information is generated on top of it. In general, we can distinguish two types of awareness models commonly used in existing awareness systems: *space-based* and *event-based* models.

3.2.1.1 Space-based models

Many researchers have previously described systems to promote awareness based on the spatial metaphor [8, 85, 91, 101]. These space-based models explicitly represent the field of work as objects (which might represent actors, information, resources, or other computer artifacts) situated and manipulable in some space. Awareness is then achieved by the interaction between actors within the space, as they present themselves and attend to each other in the field [85]. Awareness information thus is implicitly embedded as perceivable properties of objects in the space.

The use of space-based model relies on presenting a 'shared space' among collaborators at any given time to provide awareness information, both implicitly and explicitly [28]. By maintaining the state of the shared work setting, people

can either keep an eye on what the rest of the group is doing while doing their individual work, or actively monitor the activities of others, to perceive awareness information [101].

In existing awareness systems, the 'shared space' can take different forms [5].

1. 'Shared physical space'. In the early work of supporting awareness, a significant effort was devoted to exploring the potential of media space technologies, i.e. an array of continually audio-video links between distributed actors, to provide a 'shared physical space' between actors in different locations [29]. As with the awareness study, the media space investigations were concerned with informal or social aspect of awareness, and in most cases task-oriented awareness was not mentioned explicitly [93].

2. 'Shared virtual space'. Rodden [85] developed the notion of 'virtual space' as a collection of computer-supported interactive spaces. Many collaborative systems offer various types of virtual spaces to support awareness, including abstract media space [75], virtual meeting rooms [12], and collaborative virtual environment [9]. Unlike traditional media spaces that aim to establish the 'shared physical space', this line of work focuses on the use of abstract representations as awareness indicators. Besides providing a kind of 'shielding' for privacy of the people in the shared space [75], a more interesting feature of using abstract representations is the capability to organize awareness information around task-oriented structures, such as office tables, meeting rooms [12], so that more task-oriented aspects of awareness can be supported.

3. 'Shared workspace'. 'Shared workspace' is a specialization of the notion of 'shared space' that emphasizes the task-oriented awareness. According to Gutwin and Greenberg [49], a 'shared workspace' is a bounded space where people can see and manipulate artifacts related to their activities. A group editor is a good example of this type of 'shared workspace', as it serves to organize activities like writing and revising, while maintaining a coherent view of the whole [28]. The awareness information in 'shared workspace' is presented by attaching it to the manipulable objects through which the task is carried out, and is achieved by user's interaction with the artifacts [49].

3.2.1.2 Event-based models

Event-based models, on the other hand, provide people with awareness of what is going on around them as expressed by discrete events [83]. Each event highlights a piece of awareness information related to certain state change in a collaborative setting for a limited amount of time. Events can be generated by sensors that are associated with actors, shared material, or any other objects that constitute or influence a cooperative environment [78]. The list of events that are supported in each awareness system can be totally different, depending on the application domain. As argued by Fuchs et al [43], we can basically imagine any kind of event, as long as it has a certain relevance when it comes to coordinating the work in a given setting.

Unlike space-based models where the awareness information is implicitly embedded as properties of objects in the underlying representation of the field of world, awareness information is explicitly represented as first-class objects in event-based models. Each event contains identifiers of the originator, the time, the state change in the collaborative setting, and the context in which the state change took place [42]. Modeling awareness information as first-class objects decouples it from the representation of field of work, which allows the awareness information refers to any change in the field of work, but not necessarily within the user's current viewpoint in the field. The user may attend to their own individual work in the filed, but still be able to receive events about other users that are outside his/her current focus. In this way, the user does not need to monitor the field of work, instead the awareness information is pushed to the user when it becomes relevant.

Because of the decoupling of awareness information from the representation of field of work, many existing event-based systems actually do not need to provide an explicit representation of the field of work [78, 39]. Even in some systems where the field of work is represented (for example, the GroupDesk [43] and the POLIAwaC systems [103] use semantic networks comprised of actors, artifacts, events, and their relations; the BSBW system [11] and Atmosphere model use hierarchical structures of workspaces, to represent the field of work), they are mainly used by the designers to specify events and manage event distributions.

Table 3.2 shows a summary of the major distinguishing features of the *space-based* and *event-based* models. In the following of this section, we will discuss

details on how the design aspects in various cognitive and social processes as shown in Section 3.1 have (or have not) been supported each of these models.

	Space-based models	**Event-based models**
Field of work	explicitly represented as a 'shared space',	implementation-dependent representations,
	visible to users	invisible to users
Awareness information	embedded as perceivable properties of objects in the space	explicitly represented as discrete events
	presented in the context of its origin	can be outside the user's current viewport
	users need to pull the awareness information from the field of work	awareness informaion is pushed to the users
Example Implementations	1. physical spaces (e.g. Portholes [29]) 2. virtual spaces (e.g. AROMA [75], DIVA [12], MASSIVE-2 [9]) 3. workspaces (e.g. ShrEdit [28], GroupKit [86])	GroupDesk [43], POLIAwaC [103], NESSIE [78], Elvin [39]

Table 3.2: The main distinguishing features of space-based and event-based models

3.2.2 Awareness processes in space-based models

3.2.2.1 Support for perception

Selection The first step to support perception regards the selection of the awareness information for a particular user, i.e. what awareness information should be presented? Systems adopting the *space-based* awareness model usually depend on the various relationships between objects inhabiting in the shared space to decide on this.

Benford et al.'s spatial model [8] provides a formal approach to managing awareness levels between objects inhabiting in a common spatial frame. Awareness between objects is manipulated via *focus* and *nimbus*, two subspaces within which an object chooses to direct either its presence or its attention. Then the aware-

ness information is selected based on the overlapping between the receiver's *focus* and the performer's *nimbus* in certain medium [8]. Anything about the performer happens in the overlapping area will be perceivable to the receiver.

The original spatial model has been extended in several ways to support different awareness systems. Sandor et al. [91] redefined the concepts of *focus* and *nimbus* upon a semantic network that forms a representation of the working context, and use them to build a generic model 'Aether' for supporting awareness in collaborative systems. Rodden [85] adopted an object-oriented approach to generalize the spatial model to provide notions of presence, sharing and awareness applicable to applications lacking a spatial metaphor. Simon and Bandini [101] based on the spatial mode to propose the reaction-diffusion model that can support more user adaptation and dynamic features. Instead of using *focus* and *nimbus*, the awareness information is distributed using field and sensitivity function that allow for a more flexible and compact way of representing various types of nimbi and foci.

Although various space-based models differ from each other in the specific calculation of awareness levels, the most important point emphasized in all of this work is the insistence that the selection of awareness information is a joint-product between the performer and the receiver, i.e. how the receiver directs the attention to the performer (*focus*) and how the performer projects the presence or activity to the receiver (*nimbus*).

Presentation Presenting the awareness information to the user in space-based systems can be achieved in two ways: presenting the information in place of its origin, or presenting in dedicated awareness widgets [86].

As the awareness information is implicitly embedded as properties of objects in the shared spaces, it is natural to present the awareness information directly in place of the associated object in the general presentation of the field of work. For example, in the DIVA shared workspace [12], the color of document icons indicates the document status (e.g. green means 'modified by others'). As argued by Dourish and Bellotti [28], presenting awareness information in place prevents users from having to switch their attention focus between different information sources. However, this approach relies on the user's capability to perceive and

retrieve information from the shared space, which cannot always be guaranteed [12].

On the other hand, some space-based systems have also explored more lightweight awareness widgets for presenting awareness information [50]. For example, DIVA uses a virtual office table to display participants in a chat room, as well as visualizing their contributions over time[12]. The Babble system [34] includes a 'social proxy' showing team member's level of contribution to a threaded discussion. However, this approach has the major drawback that there is no direct connection between the awareness information and the shared space, which can be disturbing when the user has to switch the focus back forth between the different views. As a result, these lightweight awareness widgets are usually more beneficial for displaying specific aspects of social awareness, such as the collaborator's arrival, availability, involvement, but cannot adequately support task-oriented awareness [20].

3.2.2.2 Support for comprehension

Comparing with event-based systems, support for comprehension can be achieved relatively effortlessly in space-based systems because of two features: the existence of an explicit representation of the shared space, and the possibility of presenting awareness information directly in the shared space. Most existing space-based systems provide visual-spatial displays [52] of the shared spaces, which has a number of advantages to support comprehension. First, they organize information by indexing it spatially. In these displays, space in the display represents space in the field of work, so that if the representation of two items is close in the display, it is likely that those items are also close in the represented field of work. Therefore, information that needs to be related in interpreting and making inferences is likely to be represented by visual features that are close in the display [52]. Second, they provide overviews of the whole field of work, and therefore can be perceived as a whole and provides necessary background for comprehending awareness information [12]. Furthermore, as the awareness information can be directly presented in the place of its origin, the user can offload the cognitive effort to locate the related objects onto perceptual processes [63].

However, space-based systems also have limitations in supporting comprehen-

sion. First, we need to distinguish the difference between the context of *origin* of the awareness information and the context of *work* of the user who need to comprehend the information [45]. Most space-based systems present the awareness information in place of the object where the information occurs on, i.e. within the context of its origin. However, what is more important in the comprehension process is for the user to understand the effects of the awareness information on his/her own work, i.e. within the context of the user's work. Second, the visual-spatial display of the whole field of work can become a difficult or even impractical task when the complexity of the collaboration scales up. When the number of actors, artifacts, and activities increases significantly, such a representation of the whole shared space will also become less efficient to support the comprehension.

3.2.2.3 Support for projection

To our knowledge, explicit support for projection is rarely discussed in existing space-based awareness systems. Most of the space-based systems focus on representing the current state of the 'shared space', but how new awareness information will lead to future changes in the 'shared space' is usually considered as a cognitive task that is performed by human users. In addition, space-based systems represent the field of work using the spatial metaphor, i.e. the objects are connected based on their spatial relationships in the space. The dependency relationships, which are important knowledge to predict future changes, are not represented.

3.2.2.4 Support for feedthrough

In space-based systems, the process of feedthrough, i.e. making consequences of individual activities apparent to other participants, can be achieved either through the direct video-audio links between actors [29], or through the artifact mediation [107].

In media space-based systems, feedthrough can be supported by the networks of audio and video to provide rich representation of people and their immediate surroundings [29]. By seeing others through the media space, it is believed that people can get a sense of others activities so that they can infer the consequences of their activities. However, Fish et al.s study in the use of the Cruiser media space

[38] has shown that even though the media space allows the users to perceive each other's activities, it usually does not provide the visibility of the work artifacts involved in these activities. As a result, showing each other's activities does not necessarily mean the consequences of their activities can be fed through.

As a result, a more common approach to supporting feedthrough in space-based systems is through the common artifacts in the shared spaces, especially in shared virtual spaces and workspace [12]. In these systems, users' activities are organized around the common artifacts in the shared spaces. The artifacts provide the awareness information in how they manifest themselves within the space, and in how they provide the group with feedthrough, i.e., when artifacts are manipulated in some activities, they give off information that informs others the consequences of these activities [107]. However, artifact-mediated feedthrough requires the objects of user's activities can be represented as some artifacts, which may not be applicable in many real world collaboration. For example, the activity of police officers patrolling along the street cannot be easily organized around artifacts.

3.2.2.5 Support for manifestation

The basic way to support manifestation in space-based systems is annotations, which are defined as hypertext nodes that are linked to the objects in the shared spaces [121, 115]. Annotations allow the users to add their comments or interpretations on specific objects or locations in the shared space that are visible to other collaborators. In this way, annotations provide a way for the users to externalize their individual awareness and display them to others. However, annotations in space-based systems are often bound to the objects in the shared space, and therefore have limited capability to manifest other aspects of individual awareness, such as the intentions to perform activities, or state changes of certain activities.

Another more sophisticated way to support manifestation has been integrated in the MoMA system in the term of add-on awareness [101]. The system allows the user to specify field parameters in rule premises to construct add-on awareness promotion. For example, the user can define awareness behavior of the kind: if I receive specific pieces of awareness information, then I will trigger certain interpretation on it, and make the interpretation visible to other actors. The supported

manifestation in this way is a reactive process, i.e. actors react to certain changes in the space, however they cannot actively initiate the manifestation process.

3.2.3 Awareness processes in event-based models

3.2.3.1 Support for perception

Selection The selection of awareness information in event-based systems focuses on filter out the amount of irrelevant events presented to the users. Many mechanisms that enable the filtering of events have been discussed in the literature, and they vary from each other depending on how the filters are defined.

describe the different terms: topic-based, type-based, content-based, etc. so that they can be directly used in later chapter

In the GroupDesk system [43], Fuchs et al. define several filters that allowed the limit of the flow of events. On the actors side there was an individual privacy filter that allowed the actor to set privacy policies for the events gathered about him. On the perceivers side an individual interest filter allowed the perceiver to subscribe only to the events in which he or she was interested. A global filter would allow for the filtering of general conditions (e.g., to comply with organizational policies). Definition of filters in the system is based on individual event types, which consist of a mapping from a set of event types to a list of interested users. The drawback of event type-based filtering is that it requires the user to explicit his/her interest on each event type in a predetermined way, which is a tedious task that the users are not always willing to perform [48]. Furthermore, the type-based subscriptions tend to be too rigid or unable to adapt to the team development [2].

Alternatively, Elvin [39] adopts the content-based filtering of events. It describes events using a set of named attributes of simple data types and consumers subscribe to sets of events using boolean subscription expressions. When a notification is received at the Elvin server from a producer, it is compared to the consumers registered subscription expressions and forwarded to those whose expressions it satisfies. A key benefit of content-based notification is the capability to reduce the effort needed to manage event subscription. Instead of subscribing to each type of events, users can subscribe to event patterns, which can describe a set of events. In addition, content-based subscriptions allow the definition of composite events.

ENI [45] extends the content-based filtering approach and integrates the notion of 'contexts' into the model. Context information includes locations, artifacts and applications and other information, which is linked to a specific context. ENI adds this information to existing event information in an awareness system. The model is based on two concepts of context. First, the model tries to determine the context of origin for each event. The authors suggest a context mapping mechanism that maps events gathered from sensor information against rules saved in a context database. Second, the model identifies the work context of the user who is receiving the notification. The work context is derived from the selection of shared workspaces. Based on the two types of context, then the filtering is performed on context matching, i.e. the user defines what types of event context he/she wants to receive in a specific work context. The context-based model tries to improve the event filtering by gathering additional information and allowing users to receive awareness information in a more context-specific manner. However, the context mapping mechanisms underlying this concept is highly complex, and it lacks a formal model to specify the two types of context.

Presentation Unlike the *space-based* that relies on the users to monitor the shared space and perceive the presented awareness information, event-based systems make the awareness cues more perceivable for the users than other elements in the environment through different kinds of notification mechanisms [65]. The notification can be done in different ways. In GroupDesk [43] and POLIAwaC [103], different urgency levels are defined to determine the form of presentation of event information at the user interface. A high urgency would typically lead to a disruptive notification, such as popping up a message window, whereas a low urgency could reflect the information by a change of color of the object's icon and leave the details of information to explicit user request. In NESSIE, the presentation of awareness information is performed by configurable indicators, including simple windows for the listing of event information, different background images, or sounds. Tools are offered that allow the users to easily map awareness events to suitable indicators [78].

3.2.3.2 Support for comprehension

Comparing with space-based systems, the comprehension of events provides a more challenging task for the users, as the awareness information is typically presented in separate event-triggered displays peripheral to a person's current task-oriented concern [20]. Moreover, event-based systems often do not provide explicit representation of the field of work. As a result, in order for the user to comprehend an event, he/she has to build and maintain a mental model of the inferential framework derived from the field of work, which increases the cognitive load for the user.

To alleviate this problem, some event-based systems attempt to collocate event notifications within the context of work. For example, NESSIE provides the awareness information by the presentation of pictorial activity indicators for the members of a group as part of the shared workspace user interface [78]. In the virtual school project, Carroll et al. suggest that the deadlines and external events should be collocated with process and planning representations [20]. However, studies on supporting comprehension of events are still very limited. Questions such as what types of events should be collocated with what types of contextual representations are largely undetermined.

3.2.3.3 Support for projection

Similar to space-based systems, explicit support for projection is rarely discussed in existing event-based awareness systems as well. Most systems assume this as a cognitive task that is performed by human users.

3.2.3.4 Support for feedthrough

In existing event-based systems, the process of feedthrough can be supported by a specific type of events, i.e. activity events [43]. Activity events are generated by the system. Each time the state of an object changes due to some action of a user, a new event is generated to describe the change. Then the event is sent to the event processing server, and is distributed to the users who subscribe to it.

3.2.3.5 Support for manifestation

Rittenbruch et al. introduce and explore the notion of "intentionally enriched awareness" [84], which refers to the process of actively engaging users in the awareness process by enabling them to express intentions. They situate the "intentionally enriched awareness" between the event-based awareness systems where actors are not involved at all, and the communication tools where high level of effort from actors is required. This is very similar to our alignment of supporting the three basic awareness propagation processes, and the development of "intentionally enriched awareness" can be considered as part of the manifestation process, as it emphasizes the role of actors to make some of their internal states (intentions, reasons, etc.) along with their activities visible to others. Their AnyBiff prototypical system is one of the limited existing attempts to support *manifestation* in awareness systems, which allows users to generate, share, and use a multitude of activity indicators to reveal intentions. The AnyBiff system is mainly used for supporting social awareness in relatively loose-coupled collaborative activities, and hence the information that the users can make visible to each other is very limited. In complex collaborative environments that we are interested in this study, we believe much more aspects of individual awareness could be made visible by the users. Supporting manifestation involves not only help users express their intentions, but also make their interpretations of awareness information during comprehension/projection, or the results of their decision-making visible.

3.2.4 Summary

Table 3.2 shows a comparison of how the major design aspects of supporting awareness processes have been achieved in the *space-based* and *event-based* models.

Awareness Processes	**Space-based models**	**Event-based models**
Percepetion - Selection	1. intersections of focus and nimbus that are defined in various forms [85, 91, 8] 2. comparison of field values agaist sensitivity functions [101]	1. type-based filtering [43] 2. content-based filtering [39] 3. context-based filtering [45]

Perception - Presentation	1. in-place presentation [12] 2. awareness widgets [50, 34]	1. urgency-based notifications [43, 103] 2. configurable indicators [78]
Comprehension	1. visualization of shared spaces 2. direct linking of awaress information to the context of its origin	Collocation event notifications into the display and control of work objects [78, 20].
Projection	rarely supported	rarely supported
Feedthrough	1. video-audio links [29] 2. artifacts mediated [107]	Activity event generation and distribution [43]
Communication	separately supported	separately supported
Manifestation	1. annotations [121, 115] 2. add-on awareness [101]	intention-enriched awareness [84]

Table 3.3: Existing Studies in Awareness Support

Existing systems follow these two types of awareness models have their own pros and cons in supporting awareness processes.

Space-based models have the advantage that the awareness information is presented in the context of its origin, i.e. the associated object in the shared space, and therefore ease the comprehension process. They relies on the users to monitor the field of work and perceive the awareness information. This can be done efficiently if the local scopes of different users are largely overlapped, as the users can pick up the awareness information peripherally as they work on their own work [93]. However, this becomes much more problematic when the activities of users are distributed and each of them is working on a different set of activities, as actively monitoring the shared space requires extra attentions and efforts from the users.

On the other hand, event-based models provides a more lightweight way to present awareness information, as only the aspects of awareness information that is of relevance are presented. Furthermore, the event-based presentation is pushed to the users by making the events more perceivable to the users. In this way, the users do not need to switch their attentions to others activities until the event notification happens. However, awareness information in the form of events is usually presented

separately from the inferential framework to comprehend it, which leads to extra effort in the comprehension process.

3.3 Discussion

The major purpose of reviewing existing awareness systems in Section 3.2 is not to provide an exhaustive evaluation on existing studies. Instead, we use the review to explore design challenges in supporting awareness in complex, distributed geo-collaborative activities that will motivate our approach in the following chapters.

In general, we can identify two major design challenges based on the analysis in Section 3.2.

1. The challenge of scaling up.

Existing systems to support awareness processes are usually designed to support collaborative activities at relatively small and medium scales, it becomes a much more difficult task to support awareness in complex real time activities as we are interested in this study. As we described in Chapter 1, the collaborative activities we consider in this paper are a subset of these complex collaborative setting with two major characteristics: (1) *high level of complexity*: a large number of team workers are geographically distributed in different locations, yet engaged in interdependent activities that require effective coordination. (2) *high level of dynamics*: the actors work in dynamic settings that entail rapid changes in environment and activity, and as a result their specific awareness needs keep changing as the activities are developed.

The two awareness models (*space-based* and *event-based* models) have their own strengths and drawbacks when the collaborative activities scale up.

On the one hand, *space-based* models manage the awareness through the interaction of collaborators and rely on the users to monitor the field of work and perceive the awareness information. This provides more flexibility to handle increased level of dynamics. As the whole field of work as a shared space is explicitly visible to each user, they can pick up the awareness information relevant to them even when their own interests have been changed. However, space-based models become much more problematic when the level of complexity in the field of work increases. When the number of objects, actors and their activities is significantly

increased, the whole field of work will becomes extremely large. Representing the whole field of work and relying on the users to monitor and retrieve awareness information from it becomes a challenging task, as it requires a lot of extra attentions and efforts from the users.

On the other hand, *event-based models* provides a more lightweight way to present awareness information, as only the aspects of awareness information that is of relevance are presented. Furthermore, the event-based presentation is pushed to the users by making the events more perceivable to the users. In this way, the users do not need to switch their attentions to other's activities until the event notification happens. As a result, event-based models can handle more complex situations than space-based models. However, the effectiveness of event-based models largely depends on the quality of event subscriptions, which often requires that considerable domain knowledge be explicitly embedded. As argued by Fuchs et al. [42], event-based systems seem to work satisfactorily for situations where workflow can be clearly defined in advance or if the application is known from the beginning. When the level of dynamics increases in collaborative activities, such condition can no loner hold. The user's awareness needs are often in the flux of changes, as their activities evolve. Hence, the event-based models becomes less effective in high level of dynamics.

The above analysis clearly shows that neither space-based nor event-based models in existing studies can effectively support awareness processes in collaborative activities when both the complexity and dynamics scale up. Therefore, a new awareness model that can leverage the strengths of both space-based and event-based models becomes extremely important.

2. The challenge of integrated support.

As we conceptualize the awareness phenomena in complex, distributed collaborative activities as a continuous development of awareness knowledge through a variety of integrated cognitive and social processes, it is very important that all the individual and team processes are well supported. However, the review of existing awareness systems shows that some awareness processes have very limited support in existing studies.

At the individual level, existing awareness systems have focused on supporting perception, the higher level of awareness processes are relatively less supported. A

possible reason is that the higher-level awareness processes usually involve sophisticated cognitive and reasoning capabilities where the human can do better than the computer. As a result, most awareness systems leave the comprehension and projection to the users. However, in complex collaborative activities, as we discussed earlier, the scaling problem becomes significant, and hence it becomes much more important for the computer to provide functions to amplify and enhance human cognition, not only in the stage of perception, but also in comprehension and projection.

At the team level, existing awareness systems have focused on either supporting the explicit communication among human collaborators, or different approaches to providing 'shared spaces' representing the fields of work so that the actions of each other become visible to each other. However, less discussion has been given to support the awareness interaction via mutual displaying and monitoring, i.e. the manifestation process, which actually is an even more important aspect of awareness in many complex, real world collaborative activities [51].

As a result, the second design challenge in supporting awareness in complex collaborative activities is to consider the awareness support from a collective perspective and provide integrated support for the whole awareness development cycle.

Chapter 4

Overview of Our Approach

Our approach to supporting awareness in complex, distributed geo-collaborative activities aims to address the two major design challenges identified in Section 3.3:

1. As neither space-based nor event-based models in existing studies can effectively support awareness processes in collaborative activities when both the complexity and dynamics scale up, a new awareness model that can leverage the strengths of both space-based and event-based models becomes extremely important.

2. By conceptualizing the awareness phenomena in complex, distributed collaborative activities as a continuous development of awareness knowledge through a variety of integrated cognitive and social processes, it is very important to consider the awareness support from a collective perspective and provide integrated support for the whole awareness development cycle.

This chapter provides an overview of our approach. The general design principle of our approach is to emphasize the active role of computer system to not merely support, but rather promote awareness in complex collaborative activities. In the following, we first describe the design concept of awareness promotion, and then describe the major components of our awareness promotion framework.

4.1 From Awareness Support to Awareness Promotion

In order to address the design challenges of scaling up and integrated support, we argue that the computer system needs to play an active role to promote awareness among collaborators. Our awareness promotion approach has its basis in two basic parent disciplines: artificial intelligence (AI) and human-computer interaction (HCI).

1. From AI, we consider the computer as *an active agent* [16], situated within the collaborative environment that adaptively sense, acts, and reacts within this environment over time, to pursue its goal of promoting awareness.

2. From HCI, we act on the premise that computers and humans have fundamentally asymmetric abilities [27]; therefore, we focus on developing divisions of responsibility that exploit the strengths and overcome the weaknesses of both, and utilizing interaction techniques to facilitate effective *human-computer collaboration* [108].

4.1.1 The role of computer: from tool to actor

In existing awareness systems, the role of computer can be described in the metaphor of 'tool'. The 'tool' perspective emphasizes the human achieves his/her goal through the computer application [14]. In term of supporting awareness, it means the human actors use the computer to achieve awareness. The tool perspective emphasizes the control on the human user's side, and how the computer is used to achieve awareness depends on the human user's own knowledge. For example, in space-based systems, the system relies on the user to monitor the shared space to perceive awareness information; in event-based systems, the user needs to explicitly express his/her interests so that the computer can filter out events.

However, when the collaborative activities become more and more distributed and complicated, more is demanded of the user to control the computer. As we conceptualize the awareness phenomena as a distributed cognitive system (Section 2.4), each user usually only have partial knowledge about the whole collaborative

situation, and it is unlikely that the user will have the full knowledge of what exactly should be done in order to achieve awareness as a group. The user may have no idea what background information should be attended to in order to interpret a piece of awareness information that is out of his/her local scope of work. Or he/she cannot decide on who should be notified when his/her activities are changed. As a result, the user either uses the partial knowledge to develop awareness that is often incomplete, vague, or even incorrect; or has to extend the mental model to represent more knowledge that out of his/her local scope of work, which inevitably increases the user's cognitive load.

As a result, we believe that merely relying on the user to control the computer as a 'tool' to achieve awareness in distributed, complex collaboration is ineffective. Alternatively, a better approach is to allow the computer to take some control away from the user and actively perform some tasks to help the user to achieve awareness. Instead of relying on the user's internal, partial knowledge to control the awareness development, the computer can maintain a much more complete knowledge representation of the whole field of work at the systematic level, and make use of this knowledge to infer the user's need in the various awareness processes so as to provide awareness support. In this way, the computer can be considered as a 'mediator' actively engaged in the collaborative activities along with human actors, but with its own goal (i.e. to assist human actors to achieve awareness), its own mental model (i.e. the knowledge representation of the whole field of work), and its own behaviors (i.e. the capabilities to reason about the user's need in the various awareness processes and adapt interaction and information presentation techniques to support it).

4.1.2 Human computer collaboration

While considering the computer as an active actor to support awareness, it does not mean we can delegate all the tasks to the computer and build a completely automatic system. Existing studies in HCI and cognitive science have suggested that the relationship between human users and artificial intelligence should be treated as joint cognitive systems [27] or human-computer collaboration systems [108], emphasizing the appropriate partition of responsibility between human ac-

tors and computer systems. This perspective of human computer interaction is based on two premises.

1. It begins with the premise that computers and humans have fundamentally asymmetric abilities. The computer's strength lies in its ability to store more information than can be stored in the human's short-term memory, to perform faster data retrieval and processing, and to conduct more reliable low-level routine inferences [16]. On the other hand, the human's strength lies in the ability to integrate information from multiple sources to provide insight necessary to draw complex, higher level inferences, and the capability to handle unexpected situations with the help of long-term memory, experiences, and even intuition. As a result, the goal of computer support is actually to investigate the relationship between the cognitive characteristics of the human and the cognitive characteristics of the computer, and get the computer to complement the human [27].

2. It assumes that the human and the computer be cooperative to each other in the interaction. It means not only the computer will keep track of the human's activities and adapt its behaviors to support human performance, but also the human is willing to exposing his/her goals and activities to the computer, helping system maintain the knowledge representation, giving away control to the computer, and modifying the computer's behaviors when necessary [108]. The goal is not to shield users from the complexities of interaction, but rather to focus on the use of interaction techniques to facilitate effective human-computer collaboration.

In our approach, we also base on these two premises to propose supporting the awareness development in distributed, complex collaboration as a human computer collaborative system. While human actors still need to undergo the cognitive processes to develop individual awareness, and use it to make decision and perform activities in their own local scopes; the computer system take the responsibility to maintain a collective picture of the whole field of work, and utilize this knowledge to facilitate the various cognitive and social awareness processes among human actors.

In sum, we use the term 'awareness promotion' to define the paradigm of awareness support that follow these two design principles: (1) the computer plays the role as an 'actor' actively engaged in the collaborative activities along with human actors, maintains a collective knowledge representation of the whole field of work, and uses it to adapt behaviors to support the various awareness processes; (2) the computer provides adequate interaction techniques to allow the human actors to collaborate with it to develop awareness at both the individual and team level.

Table 4.1 illustrates the major differences between the traditional awareness support paradigm and awareness promotion in addressing the different awareness processes. From the table, we can clearly see some of the distinguishing characteristics of awareness promotion.

1. First, the awareness promotion approach emphasizes the computer's knowledge representation of the field of work, and its relationship to awareness information. One one hand, the awareness information is used to update the computer's knowledge representation so that it matches the current state of the field of work. On the other hand, the knowledge representation is used by the computer in almost every awareness process to perform a variety of reasoning tasks to support awareness.

2. Second, comparing with the traditional awareness support, the awareness promotion approach shows much more frequent interactions between the computer and the human users. Each awareness process involves both the computer's reasoning and the human's cognition, as well as the interaction to combine them together.

Awareness processes	**Awareness support**	**Awareness promotion**

Percepetion	*Space-based* models: • the *computer* shares information in a common space • the *human* monitors the shared space to perceive information *Event-based* models: • the *human* subscribes to events based on interests • the *computer* filters out events based on human subscription	• the *computer* uses the information to update its knowlege representation • the *computer* uses its knowledge representation to infer who the information is relevant to, and present it only to the relevant actors • the *user* can modify the computer's knowledge representation
Comprehension	*Space-based* and *event-based* models: • the *computer* links the awareness information to the context of its origin • the *human* infers the connection between the context of origin and his/her work context	• the *computer* connects the awareness information to the human's work context • the *human* interprets the information, and sends the result to the computer • the *computer* updates its knowledge representaton based on human interpretation

Projection	*Space-based* and *event-based* models: • the *human* performs the projection on the own	• the *computer* performs the projection based on its knowledge representation • the *computer* presents the projection results to the human • the *human* reviews and modifies the computer's reasoning • the *computer* updates its knowledge representation based on human modification
Feedthrough	*Space-based* models: • the *computer* broadcasts the effect to everyone *Event-based* models: • the *computer* notified the effect to all subscribers	• the *computer* uses the effect to update its knowlege representation • the *computer* uses its knowledge representation to decide on who should receive the effect and notify them • the *human* can control who can see the effects of his/her activities

Manifestation	*Space-based* models: • the *human* creates an annotation • the *computer* makes it visible to everyone *Event-based* models: • the *human* indicates his/her intention as an event • the *computer* notified the event to all subscribers	• the *human* generates awareness information indicating some aspects of his/her individual awareness • the *human* can control who can see the generated awareness information • the *computer* uses the generated awareness information to update its knowlege representation • the *computer* uses its knowledge representation to decide on who should receive the information and notify them

Table 4.1: Awareness support v.s. awareness promotion

We believe these distinguishing characteristics of awareness promotion approach provide the potential to address the two design challenges, i.e. handling the scaled up complexity and dynamics, and providing integrated awareness support. First, the awareness v approach utilizes the computational knowledge representation to model the field of work and offloads some of the representation and reasoning efforts from the human to the computer. Hence, it can handle more complex situations than existing awareness models. Meanwhile, the knowledge representation is dynamically updated to reflect the current state of the field of work, which allows it to handle increased level of dynamics. Second, as the computer is equipped with the computational knowledge representation and reasoning capabilities, it allows the computer to provide support on the higher level of awareness processes.

4.2 The Awareness Promotion Framework

To operationalize the awareness promotion approach, we propose a computational awareness promotion framework. Following our conceptualization of the awareness phenomena in Section 2.4, our approach is built on top of two major knowledge components: a computational representation of the field of work based on the SharedPlan theory, and an event-driven model of the awareness processes. Then the computer system's behaviors to promote awareness are embedded in the interaction between these two components. On one hand is how the computer constructs and develops the knowledge representation of the field of work within the event-driven processes, and on the other hand is how the knowledge representation is used to promote these event-driven awareness processes.

This section provides an overview of the awareness promotion framework and discusses the major design choices, but leave the details to the next few chapters.

4.2.1 Computational representation of the field of work

The nature of awareness phenomena in distributed, complex collaboration as we conceptualized in Section 2.4, and the goal of promoting awareness impose several requirements for the knowledge representation of the field of work:

1. The knowledge representation should provides a precise model that can capture knowledge about all the three constructs that compose the field of work, i.e. the basic elements of human activities, the local scope of work for each actor, and the various dependency relationships among the activities.

2. The knowledge representation should be dynamically updated. Since the awareness needs of users change quickly with the field of work in distributed, complex collaboration, it is essential that the knowledge representation should be kept updating so that it always reflects the current state of the changing field of work.

3. The knowledge representation needs to be formalized and can be computationally modeled so as to support computer reasoning.

Existing awareness systems have provided several approaches to representing the field of work. The AREA system [42] describes the field of work as semantic networks including relationships among objects, where objects can be human actors artifacts, or aggregations such as groups of people. The representation is primarily used for the users to specify which objects and associated events they are interested in and when they want to be informed. The Atmosphere model [82] describes the field of work as a hierarchically structured workspace that consists of a set of *spheres* and *sub-spheres*. Users classify their actions on artifacts by mapping them into different spheres. The MoMA model [101] applies a reaction-diffusion metaphor to model the field of work as a set of entities embedded in an interaction space, which behave by using diffusion and reaction capabilities. This metaphor is based on the idea that whenever two or more entities have contact, their states are modified in some way. Their states are then propagated to others through fields in the space.

Although these representations have shown their capabilities to support specific aspects of awareness in their respective application domains, none of them can meet all the three requirements for knowledge representation to enable awareness promotion. The AREA model provides a good representation of the activities and the dependencies using the semantic networks, but the local scopes of each user are not captured. The model is static (as it is pre-determined by the designers), and informal (as it is primarily used as a descriptive framework). The Atmosphere model organizes the field of work around the artifacts without explicit representation of activities. It supports the specification of each user's local scope using private 'spheres', but no dependency relationships between activities are supported. It supports the modification to the representation by human users, but the system cannot automatically adapt the representation to changing environment. The MoMA model can support all the three constructs of the field of work as the definition of entities and spaces are generic, and provide some reasoning capabilities. However, it does not support the dynamical adaptation of the model.

In this study we draw knowledge representation and reasoning techniques from existing studies in artificial intelligence to develop a computational model of the field of work that can satisfy all the three requirements. An important feature of our model is the focuses on intentions, beliefs, knowledge, and other attributes

of actors' mental states [46]. As human actors' awareness processes are directed by their mental models (Section 2.4.3.1), we believe that understanding and representing human actors' mental states allow the computer to infer each actor's local scope, and derive awareness needs from it. We will describe details of the computational representation of the field of work in Chapter **??**, and the dynamic adaptation behaviors in Chapter **??**.

4.2.2 Event driven awareness processes

In our approach, we adopt the general event-driven interaction paradigm [36] to model the awareness processes. In an event-driven mode of interaction, actors (both human and computer) communicate by generating and receiving events. Each actor receives events from environment and other actors, reacts to them, and generates new events to other actors. We believe that the event-driven paradigm is suitable for modeling awareness processes in our study for two major reasons:

emphasize the iterative process, the events are connected.

1. The collaborative environments under discussion in this study are naturally centered on events. As we focus on the set of physical collaborative activities that entails rapid and frequent changes in environment and activities, human activities are usually triggered by the detection of events, and then are performed to analyze and react to these events. In the motivating scenario of emergency response, for example, the collective efforts are triggered by an incident, or a disaster, i.e. an event happening in the environment. To respond to the initial event, actors perform activities that add more events building up the situation. In addition, unexpected new events can occur in the environment, requiring continuous response [113]. It is these critical events that form the actors' awareness needs and cause the actors to adopt certain goals and perform activities to resolve possible problems.

2. The event-driven paradigm provides an effective way to represent and process awareness information. Instead of asking the actors to monitor the environment and pull awareness information from it, events are pushed to the actors as they occur. In this way, the actors do not need to split their limited

attentions to monitor the environment and each other's activities. This is very important for supporting awareness, as one general design concern for awareness system is that awareness must be achieved with minimal attention and effort so that it does not interrupt the current line of action [93].

Our event-driven model of awareness processes shares some commonality with existing event-based models, as both use events as the basic unit to organize and present awareness information. However, they also differ in several important ways:

1. In existing event-based models, the computer primarily plays the role as producers of events. It detects events from sensors in the environment or feedbacks of human actions, filters them out based on user subscriptions, and present them to the users. However, in our approach, the computer is also the consumer of events. As to maintain the knowledge representation of the field of work, the computer needs to take the events as input of its reasoning process, and use them to make inference and update its knowledge representation. In this way, the computer behaves like other human actors to consume the events to develop its own awareness.

2. The concept of events in our approach has a much richer meaning than existing event-based models. It is not only used to describe the occurrences in the environment and in human activities, but also the psychological experience of these occurrences as human actors perceive and interpret them. We make a clear distinction between real world occurrence, event, and awareness in Section 4.2.2.1.

3. Existing event-based models focus on the generation and presentation of events to the users, but our approach emphasizes the whole process of awareness development driven by events. This means we are not only concerned about how to select and present events to the users, but also how to support the interpretation of these events, and how new events are generated based upon existing ones.

In sum, we consider the event not only as a way to represent awareness information, but also an effective interaction mode to drive the awareness development.

the support is not discrete, but continuous.

Each round of awareness development is triggered by receiving certain events, and finished with generating new events that will be consumed by other actors. In the following, we first clarify our concept of event, then describe the event-driven awareness processes.

4.2.2.1 The concept of event

Before going further we should clarify what we mean by an event. The concept of the word 'event' has been used in the literature from different perspectives:

1. The first meaning refers to an actual occurrence (the something that has happened) in the real world. Set out by Quine [79], events in this meaning are first-class entities that can be localized in space and time, broken into sub-parts, and arranged in a taxonomic hierarchy. Research using this concept of events primarily focuses on studying the internal structures of the events. For example, in [119], complex geographical phenomena, such as wildfire or precipitation, have been modeled in a hierarchy of events, processes, and states. In [4], the event-based approach has been adopted to model all the types of occurrences in movement analysis. The focus here is to derive the different event types that may occur and identify the relationships between them.

2. The second meaning takes us into the realm of computerized event processing, where the word 'event' is used to mean a programming entity that represents this occurrence [104]. Each event in this notion is a message that describes an real world occurrence by its source, location, time, and other measurable properties. A single event occurrence can be represented by many event entities, and a given event entity might capture only some of the facets of a particular event occurrence [36].

3. In event-based awareness systems, the concept of event has a more specific meaning as representing a specific type of awareness information that might be relevant to the user's work. It is usually an application-specific concept, depending on the set of awareness information that is supported by an awareness system. For instance, the AREA system [42] describes events as actions

performed on an artifact and the event classes are derived from the artifact class hierarchy and possible operations on them. The ENI system [45] describes events from the sensors associated with actors, shared artifacts, or any other objects that generate events related to them.

In this study, we use three different words to avoid the confusion when defining events:

1. We use the word *'occurrence'* to denote the real world happenings. It can be in the environment, e.g. the occurrence of a traffic accident at a location; or associated with an object, e.g. a vehicle arrives at the accident spot; or associated with human activities, e.g. the successful performance of first aid on a victim.

2. The word *'awareness'* is used to refer to the human consciousness about the occurrences, events, and their relevance to ongoing or future human activities. It emphasizes that awareness is inherently a cognitive, interpretive, and predicative concept that reflects a state of mind.

3. We use the word *'event'* to denote a computerized entity that describes a piece of awareness information that is relevant to an actor's work. On one hand, it can be the description of a real world *occurrence* by its measurable properties, e.g. the information about a traffic accident with time, location, number of victims etc. On the other hand, it can also be the externalization of an actor's *awareness* knowledge, e.g. an actor's belief that the accident will block the traffic and cause a delay on delivery of medical team.

4.2.2.2 Awareness processes

Along with the distinction between the concepts of *'occurrence'*, *'awareness'*, and *'event'* is the notion of dynamic transformations between them (Figure 4.1). The transformations are tied to different processes in awareness development.

The direct transformation between *occurrence* and *awareness* corresponds to the individual awareness development cycle that has been described in Section 2.2.2. A real world *occurrence* is transformed into *awareness* through an actor's individual awareness processes, i.e. *perception*, *comprehension*, and *projection*.

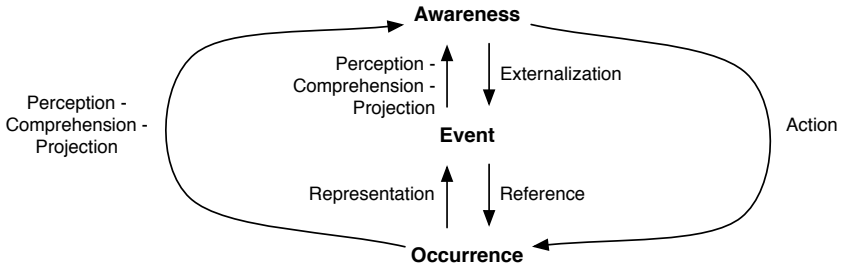


Figure 4.1. Transformations between occurrence, event, and awareness

Then, the achieved *awareness* guides the actor's *action* that may generate further *occurrences* in the real world.

The thing becomes more interesting when the computer support is involved and the transformations are driven by *events*. One one hand is a real world *occurrence* can be captured and represented as an *event* by the computer system, and presented to a human actor. Instead of perceiving the *occurrence* directly, the actor perceives the corresponding *event* in the computer interface, and develops the *awareness* upon it through the actor's individual awareness processes, i.e. *perception*, *comprehension*, and *projection*. On the other hand is some aspect of the actor's *awareness* can be externalized as a new *event*, which refers to some current *occurrence* or predicts future *occurrence* in the real world.

The event-driven transformations can also be used to describe the different awareness propagation processes across multiple actors. The process of *feedthrough* can then be described in the following steps: (1) an actor's *awareness* guides his/her *action* that generates a new *occurrence* in the real world; (2) this new *occurrence* is then captured and represented as an *event* by the computer, and presented to another actor; (3) the other actor develops the *awareness* upon receiving the new event. Similarly, the process of *manifestation* can also be described as follow: (1) an actor's *awareness* is externalized as a new *event*, which refers to some current *occurrence*; (2) this new *event* is then presented to another actor by the computer; (3)the other actor develops the *awareness* upon receiving the new event.

Figure 4.2 shows an example of how these event-driven awareness processes can be combined together to describe a complex trajectory of awareness development

that involves three actors in a group. The awareness is propagated from *Actor 1* to *Actor 2* through the process of *manifestation*, and is then propagated from *Actor 2* to *Actor 3* through the process of *feedthrough*.

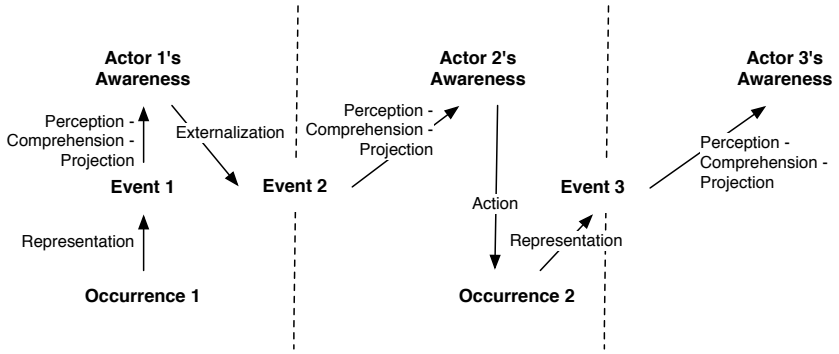


Figure 4.2. An example of event-driven awareness development trajectory

4.2.3 The framework

Built on top of the computational representation of the field of work, and the event-driven model of the awareness processes, our awareness promotion framework focuses on the interaction between these two components (Figure 4.3).

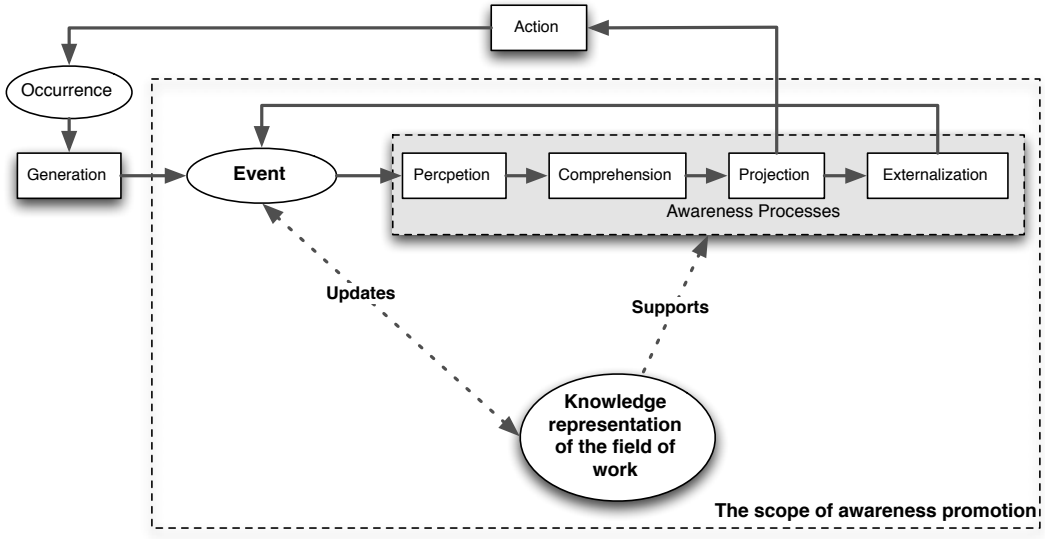


Figure 4.3. Awareness promotion framework

On one hand is how the computer constructs and develops the knowledge representation of the field of work within the event-driven processes. As we argued in Section 4.2.1, one of the requirements for the computational representation of the field of work is that it should support dynamic adaptation so that it always reflects the current state of the changing field of work. We utilize the *events* to achieve this goal. As human actors use a subset of the events to develop their individual awareness, the system processes every event to develop its knowledge representation of the whole field of work. The events can be generated by sensing the occurrences in real world, or they can be the results of human actors' externalization of their individual awareness. All of these events will be processed by the computer system to update its knowledge representation of the field of work.

On the other hand, the knowledge representation is used by the computer system to promote the awareness processes. Following the design principle of human-computer collaboration emphasizing the appropriate partition of responsibility between human actors and computer systems (Section 4.1.2), the promotion role of the computer system is to strike a balance between the system's reasoning capabilities and providing visual and interactive support. With the computational knowledge about the field of work, the system can offload some of the reasoning effort from the human actors. Meanwhile, the system knowledge can also be visualized to help the human actors perform their part of the work or overwrite the computer's behaviors.

As shown in Figure 4.3, we exclude the event generation process, i.e. the transformation from real world occurrences to events; and the action process, i.e. the transformation from awareness to real world occurrences, from the scope of awareness promotion in this study. It does not mean that these processes are less important or they can be separated from the rest. Rather, we believe these processes are relatively standalone processes that are out of the scope of this study, and therefore we do not claim contributions to these processes.

The next two chapters provide details of this awareness promotion framework. Chapter **??** describes the two knowledge components, i.e. the computational representation of the field of work and the specification of events, and the mechanisms for updating the knowledge representation with events. Chapter 6 describes the promotion of individual awareness processes and awareness propagation.

Chapter 5

Knowledge Representation and Updating

In this chapter, we describe the two major knowledge components: the computational representation of the field of work, and the events. We first provides a formalization of the field of work using a first-order logic, augmented with several modal operators and meta predicates. The formalization is not intended to be directly implemented, rather it is intended to be used as a specification for designing knowledge representation and reasoning capabilities. We then discuss how the formalization can be computational represented using the PlanGraph model. Then, we discuss the specification and typology of events in our approach.

After defining the two major knowledge components, we discuss the the knowledge updating process, which covers two important issues: on one hand is updating the knowledge representation of the field of work so that it always reflects the current state of the collaborative activity by reacting to the various external and internal events through several reasoning processes. On the other hand, these reasoning processes also enrich these events with system generated knowledge that is used to promote the human actor's awareness processes discussed in next chapter.

5.1 Formalizing the Field of Work

In general, the field of work in a collaborative activity can be defined as a domain model that includes three related sub-models $FoW = \{ER, LS, DEP\}$: the model

of basic entities and their relations ER, the set of all actors' local scopes LS, and the set of dependencies among activities DEP. ER provides the basic vocabulary for defining local scopes LS and dependencies DEP. We elaborate on the three components in the rest of this section.

5.1.1 Entities and relations

5.1.1.1 Entities

Following the conceptualization of the field of work in Section 2.4.1, we define three basic entities that are the basic building blocks in a collaborative activity:

1. **Actors** are defined as human actors participated in a collaborative activity, capable of making decisions and performing actions. $AR = \{ar_1, ar_2, ..., ar_n\}$ is the set of all the actors in a collaborative activity.

2. **Resources** are anything that can be used in the transformation process of an activity, including both material resource and resources for thinking. We use $RES = \{res_1, res_2, ..., res_n\}$ to denote the set of resources used in a collaborative activity.

3. **Actions** include all the actions that have been performed, is under execution, or will be performed in a collaborative activity. $ACT = \{a_1, a_2, ..., a_n\}$ denotes the set of actions.

We presume all the basic entities fall into types. Each *action type* represents a class of actions that can be performed by executing a set of subsidiary tasks, commonly called a *recipe*. For example, an actor might rent a car by walking to the rental car center, selecting the car, paying the money, and so forth. Each *resource type* represents a class of resource objects that share the same list of properties and behaviors. For example, every car at the rental car center has a make, model, rental price, etc. Each *actor type* represents a role that abstracts the behaviors of a social actor within some specialized context or domain of endeavor [15]. Its characteristics are easily transferable to other social actors. For example, every actor at the rental car center in the role of 'receptionist' can help the customer to rent a car.

We follow Hunsberger and Ortiz [57] to use the @ constructor to make the distinction between types and instances of entities in the formalization. Each type expression is defined as a unique keyword (e.g. $AType$), then each instance is in the form of the type keyword followed by a set of arguments necessary to identify it (e.g. $AType@arg_1(val_1)@arg_2(val_2)...@arg_n(val_n)$). For example, *drive* can be an action type and an instance could be $drive@car(car_3)@from(loc_1)@to(loc_2)$ that specifies an instance of driving car_3 from location loc_1 to loc_2. $receptionist@name(Mike)$ is an actor named $Mike$ in the role of *receptionist*. To simplify the expression, we may sometimes use a type keyword followed by a subscript to define an instance. For example, $receptionist_1$ can be used to define an instance of *receptionist*.

5.1.1.2 Relations

The relations between these entities can be defined using predicate symbols. For example, predicate symbol $Located_in$ can be used to indicate the spatial relation of one entity is inside the second entity. The fact that actor ar_1 is in the office $room_3$ can be expressed as $Located_in(ar_1, room_3)$. The list of predicate symbols and their meanings are usually domain dependent. However, here we are more interested in defining a subset of relations between these entities that reflect the common characteristics of activity structure, and are used later to define the local scopes LS and dependencies DEP.

Actor's intention towards action We follow the SharedPlans theory [46] to define three types of intention predicates, $Pot.Int$, $Int.Th$, and $Int.To$, to represent the *intentions* towards an action that have been adopted by an actor.

1. $Pot.Int$ is used to indicate that an actor would like to adopt an intention that the action be performed, but which has not yet be committed to.

2. $Int.Th$ is used to represent an agent's intention that the action be performed.

3. $Int.To$ is used to represent an agent's intention to do some action.

The major difference between these three is the level of commitment the actor has on the action. $Pot.Int(ar_1, act_1)$ indicates that actor ar_1 is considering adopting the intention that the action be performed, but has not yet committed

to it. $Int.Th(ar_1, act_1)$ indicates that actor ar_1 has intended that the action act_1 be successfully performed, but it could be performed by other actors. The actor may commit to help the other actors to get it done, or avoid conflicts, but is not participated in the performance of the action. In the proposition $Int.To(ar_1, act_1)$, the actor ar_1 has full commitment to perform action act_1, i.e. ar_1 will participate in the performance of the action act_1.

In addition to the three intention predicates, we also define the $Perform$ predicate to indicate that the actor is in the process of performing the action.

Actor's capability of performing action The *capability* of an actor to perform an action can be defined using three predicates:

1. $Knows$ is used to indicate that the actor has the knowledge about how to perform the action, i.e. the actor has at least one recipe to perform the action.

2. *Able* indicates that the actor has the ability (e.g. expertise, skills, resources) to perform the action.

3. $Workable$ indicates that the actor can actually perform the action by satisfying all the constraints and pre-conditions.

The three *capability* predicates describe different levels of capability of an actor on an action. $Knows(ar_1, act_1)$ merely says that ar_1 knows how to perform the action act_1, but may not be able to perform it. For example, a blind man may know how to drive a car, but do not have the ability to drive it. Furthermore, $Able(ar_1, act_1)$ indicates that the ar_1 has the ability to perform the action act_1, but it does not necessarily mean the actor can actually perform it. A man who is able to drive the car may be located in a different country, and hence he cannot actually drive it. $Workable(ar_1, act_1)$ defines the highest level of capability, as the ar_1 can meet all the constraints and actually perform the action act_1.

Relations between actions An important characteristic of actions is that they can be either basic or complex. A *basic* action can be directly executed. A *complex* action, however, cannot be directly executable because it needs to be decomposed

into subsidiary actions. The relations between an action and its subsidiary actions can be represented by the predicate *Sub.Act*. $Sub.Act(act_1, act_2)$ indicates that act_1 is a subsidiary action of performing act_2 within the current plan. The *Sub.Act* predicate only indicates the relationships between these two actions under the context of current plan. If the plan is changed, the relationship may no longer hold.

Besides the *decomposition* relation between actions, the actions can also be related by the *precedence* relation. The *Precedes*) relation defines the temporal order of performing two actions at the same level. For instance, $Precedes(act_1, act_2)$ indicates that act_1 should be performed before act_2.

Relations between action and resource We use two predicates, *Consumes* and *Produces* to define the relations between an action and a resource. $Consumes(act_1, res_1)$ refers to the situation that the performance of action act_1 requires the use of resource res_1. On the other hand, $Produces(act_1, res_1)$ indicates that action the performance of act_1 will produce the resource res_1, or make the resource ready for other actions.

add a table to summarize all the relations described in this section.

5.1.2 Local scopes

We define the local scopes of actors based on two types of relations between actors and actions, i.e. *intention* and *capability*, defined in Section 5.1.1.2.

1. The *intention* relations define the set of actions that an actor has intention or potential intention towards their performance. As we assume the actors' behaviors be goal-driven [69], this set of actions can be used to define the sub-space of the field of work that the actor is willing to work on, we call it *local scope of intention*.

2. The *capability* relations define the set of actions that the actor have certain level of capability to work on. This set of actions define the sub-space of the field of work that the actor can have impact on, we call it *local scope of capability*.

The whole local scope of work for an user is then defined as the union of these two sub-spaces defined by the *intention* and *capability* relations (Figure 5.1). The actions within the intersect of these two sub-spaces are these that the actor actively participate in. The actions within the *local scope of intention*, but outside the *local scope of capability* are these that the actor may need help from other actors. The actions outside the *local scope of intention*, but inside the *local scope of capability* are where the actor can offer help to other actors.

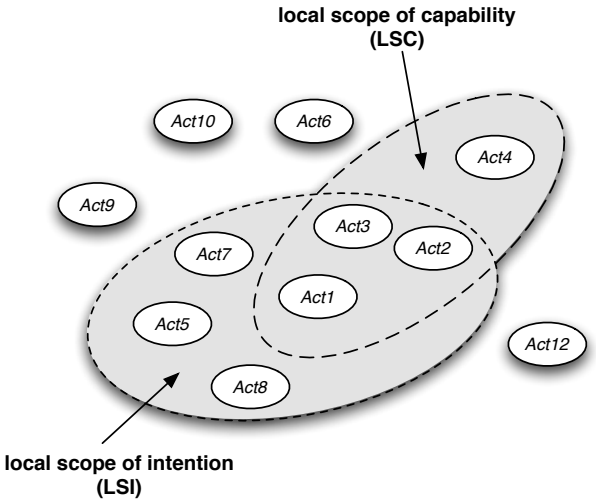


Figure 5.1. The structure of local scope of work

One thing to note is that the actions within the same sub-space can have different relations with the actor. That is, the actions in the *local scope of intention* can be either potentially intended ($Pos.Int$), intended that it be performed ($Int.Th$), or intended to be performed ($Int.To$). Similarly, the actions in the *local scope of capability* can also be related to the actor with different levels of capability (i.e. $Knows$, $Able$, $Workable$). As a result, we cannot define the local scope for each actor just as a set of actions, but also with the corresponding relations.

Following this, we define the *local scope of intention* for an actor $LSI(ar)$ as a set of tuples, and each tuple includes three elements: the actor, the action, and the intention relation between them:

$$LSI(ar) = \{< ar, act, rel(ar, act) >\}$$

where $act \in ACT$ and $rel \in \{Pos.Int, Int.Th, Int.To, Perform\}$.

Similarly, the *local scope of capability* for an actor $LSC(ar)$ is defined as:

$$LSC(ar) = \{< ar, act, rel(ar, act) >\}$$

where $act \in ACT$ and $rel \in \{Knows, Able, Workable\}$.

Thereafter, the *local scope of work* for an actor $LS(ar)$ is defined as:

$$LS(ar) = LSI(ar) \cup LSC(ar)$$

5.1.3 Dependencies

We can now focus on the modeling of dependencies between actions. In general, a dependency is defined as a meta-predicate (DEP) on two actions (act_1, act_2) and a proposition p, where act_1 depends on act_2 because of some proposition p, and is denoted by $DEP(act_1, act_2, p)$. We follow the terms used in [118] to call the depending entity act_1 the *depender*, and the entity who is depended upon act_2 the *dependee*, and the proposition representing the dependency relation *dependum*.

Based on the various types of relations between actions and resources defined in Section 5.1.1.2, we can use them to define the various types of dependencies described in our conceptual framework in Section 2.4.1.3.

Temporal dependency The predicate $Precedes$ can be used to define the temporal dependency between actions, i.e. the performance of the action act_2 as *depender* depends on the performance of the action act_1 as *dependee*, because act_1 is a pre-requisite action that must have been completed at the time when a_2 is performed. $Precedes(act_1, act_2)$ is the *dependum* in this case.

$$DEP(act_2, act_1, Precede(act_1, act_2))$$

Resource dependencies The predicates $Consumes$ and $Produces$ can be used to define the three types of resource dependencies.

A *fit dependency* occurs when two actions (act_1, act_2) collectively produce the same resource (res_1). In this case, the two actions are mutually dependent on each

other, i.e. both can be *depender* and *dependee* at the same time. The *dependum* is the combination of two predicates: $Produces(act_1, res_1)$ and $Produces(act_2, res_1)$.

$$DEP(act_1, act_2, Produces(act_1, res_1) \wedge Produces(act_2, res_1))$$
$$DEP(act_2, act_1, Produces(act_1, res_1) \wedge Produces(act_2, res_1))$$

A *flow dependency* arises whenever one action act_1 produces a resource res_1 that is used by another action act_2, i.e. the performance of the action act_2 as *depender* depends on the performance of the action act_1 as *dependee*, because of the two predicates: $Produces(act_1, res_1)$ and $Consumes(act_2, res_1)$.

$$DEP(act_2, act_1, Produces(act_1, res_1) \wedge Consumes(act_2, res_1))$$

A *sharing dependency* arises whenever multiple actions both use the same resource. Similar to a fit dependency, the two actions are mutually dependent on each other, because of the two predicates: $Consumes(act_1, res_1)$ and $Consumes(act_2, res_1)$.

$$DEP(act_1, act_2, Consumes(act_1, res_1) \wedge Consumes(act_2, res_1))$$
$$DEP(act_2, act_1, Consumes(act_1, res_1) \wedge Consumes(act_2, res_1))$$

Goal dependency The predicate *Sub.Act* can be used to define the goal decomposition dependency between two actions. It indicates that action act_2 depends on action act_1 because act_1 is a subsidiary action that must have been completed to achieve act_2 in current collaborative plan, i.e. $Sub.Act(act_1, act_2)$

$$DEP(act_2, act_1, Sub.Act(act_1, act_2))$$

5.2 Representing the Field of Work with PlanGraph model

In this section, we present the knowledge representation of the field of work based on PlanGraph model. By modeling an collaborative activity within a PlanGraph, the entities and relations in the field of work as we formalized in Section 5.1.1 can

be represented as different components of the PlanGraph. Then we show how the local scopes can be easily derived from the PlanGraph model. Last, we describe the construction of dependency network from the PlanGraph model.

5.2.1 The PlanGraph model

This part needs to be updated and revised, including what attributes are stored in each node, add condition node, allow resource has conditions; how condition is represented

The PlanGraph model is designed to represent the dynamic knowledge in the human-computer collaboration based on the SharedPlans theory [19]. The PlanGraph model has been used in several existing studies to model human-computer collaboration in different applications, e.g. the natural conversational interface to geospatial databases [19], collaborative dialog-based system to communicate vague spatial concepts [18], and context-aware mobile mapping [117].

In general, a PlanGraph represents the knowledge about a collaborative effort towards a common goal in a hierarchical way (Figure 5.2). The root of a PlanGraph is the overall goal of the actors in a collaborative activity, which is decomposed recursively as actions through the adoption of recipes. The whole PlanGraph therefore is a shared plan corresponding to the root action, while each sub-tree with a sub-action as the root represents the shared plan for that sub-action. In a PlanGraph, besides the action nodes, we also represent two types of nodes: a *parameter* represents an informational or physical object that is used by an action; and a *condition* represents a state of affairs in the world that the actors would like to achieve.

There are several ways that these three types of nodes can be connected in a PlanGraph:

1. An action can be decomposed into several parameters, conditions, and subsidiary actions, where the parameters indicate all the informational or physical objects that will be used by the action. All these parameters need to satisfy their own constraints before the action can be performed, and they are accessible by all the subsidiary actions at the same level. The conditions under an action list all the constraints that need to be satisfied before the action or any subsidiary actions can be performed.

2. Each parameter can be decomposed into subsidiary actions and conditions.

The children actions of a parameter are used to identify the value of the parameter, i.e. they are used to satisfy the knowledge-precondition on the parameter. The conditions attached to a parameter are the constraints that need to be satisfied before the parameter is ready to be used by upper action.

3. Each condition can only be decomposed into subsidiary actions that are used to achieve the condition.

The PlanGraph model also encodes the actors that are participated in each action or sub-action. Because the relations between actors and actions can be many-to-many, i.e. an actor can work on multiple actions, and one action can involve multiple actors, we represent knowledge of actors within their relations towards each action. Therefore, each action node in a PlanGraph includes several attributes to store the relations with participating actors as defined in Section 5.1.1.2: *Intentions* records the different intention relations of each actor towards the action. *Capabilities* indicates the level of capability of each actor to perform the action, such as whether they have knowledge to identify a recipe for the action; or whether the agents can bring it about.

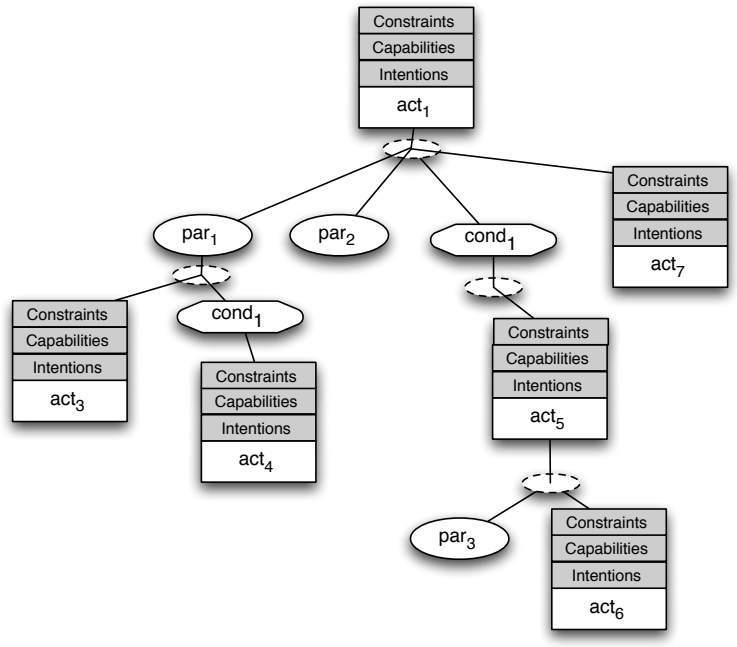


Figure 5.2. Structure of a PlanGraph

Show a concrete example of plangraph in the

In many applications, such as the dialog-based interfaces [18, 19], the whole collaborative activity can be represented using one single PlanGraph, as the overall goal in one interaction session is unique. However, in complex collaborative activities, it is possible that multiple goals are active at the same time. As a result, the knowledge representation of field of work in these situations is often a forest that includes multiple PlanGraph trees in parallel.

5.2.2 Representing elements and relations

The PlanGraph model allows us to represent the basic elements and entities in the field of work. The actions, actors, and resources are first-class objects in the PlanGraph model, and the multiple relations among then can be represented as the structural relations in the PlanGraph model (Table **??**).

Add the class diagram or the picture similar to the one used in the social modeling book to show the different entities

Actors Each actor is represented an object with a unique *ID* in the PlanGraph model. Each actor object in the PlanGraph maintain the current state of the corresponding actor, such as the name, the location, the role of the actor, and the expertise he/she has. Each actor object also maintains a list of actions that the actor is currently participated in.

Actions Actions are modeled as a type of node within the hierarchical structure of the PlanGraph. The *goal* of the action is represented as a condition node, indicating the expected effect of the action when it is successfully performed. The current *execution state* of the action and the *current recipe* to perform this action are stored as attributes of each action node. The *Intentions* and *Capabilities* record the different relations between participating actor towards the action. The *Constraints* record the possible constraint relations between subsidiary actions. The *parameters*, *conditions*, and *sub-actions* point to the other nodes that form the sub-plan of this action.

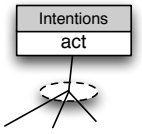
Resources Resources are modeled as parameter nodes in a PlanGraph. Each parameter node records information about the current states of the corresponding

resource in the domain, such as the location of a rescue vehicle, or the capacity of the decontamination station. Parameter nodes can be collective or individual. A collective parameter indicates a collection of resources with the same type, for instance, a parameter representing all the victims in the same rescue operation. Each resource points to at least one sub-action that is used to identify the resource, and may points to conditions that the resource needs to satisfy.

Relations between actors and their actions The relations between actors and their actions can be directly retrieved from the *Intentions* and *Capabilities* attributes attached to each action, recording the different relations between participating actor towards the action.

Relations between actions The two major types of relations between actions are composition and precedence. The former is directly encoded in the hierarchical structure of a PlanGraph. Each action node has an attribute *sub actions* recording its subsidiary actions under current plan. As a result, the fact that acton act_1 is included in the list of *sub actions* of action act_2 represents the relation that action act_1 is the subsidiary action of doing act_2. The precedence relation that reflects the temporal order of doing two actions is encoded in the *constraints* slot of their parent action node.

Relations between resources and actions The relations between resources and actions are modeled as the composition relations between action nodes and parameter nodes in a PlanGraph. Each action node has a slot *parameters* recording all the parameters under current plan, and each parameter has a slot *sub actions* recording the actions to identify the parameter, and the *conditions* that include subsidiary actions to manipulate the parameter. On the other hand, all the sub actions at the same level of the parameter are consumers of this parameter.

Relation	Structure in PlanGraph
$Pot.Int(ar, act)$ $Int.Th(ar, act)$ $Int.to(ar, act)$ $Perform(ar, act)$	search the *Intentions* slot attached to each action node: 

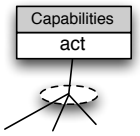
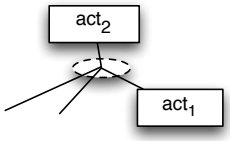
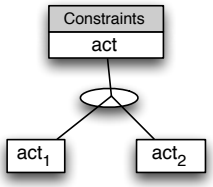
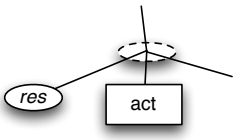
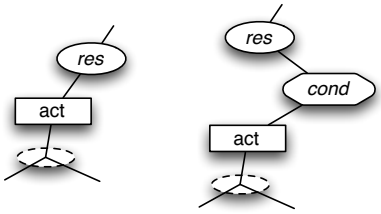
$Knows(ar, act)$ $Able(ar, act)$ $Workable(ar, act)$	search the *Capability* slot attached to each action node: 
$Sub.Act(act_1, act_2)$	act_1 is a subsidiary action node under act_2: 
$Precedes(act_1, act_2)$	act_1 and act_2 as ordered siblings: 
$Consume(act, res)$	a parameter node representing a resource res and a action node act as siblings: 
$Produces(act, res)$	act is a subsidiary action node under the parameter node representing a resource res: 

Table 5.1: Representing basic relations in PlanGraph

5.2.3 Constructing local scopes

Based on the definition of local scopes in Section 5.1.2, the local scope of an actor can be dynamically constructed by depth-first traversing through the PlanGraph model to search for all the actions that have intention or capability relations with the actor.

We define a procedure *BUILD-LS* that takes an actor object *ar* and a PlanGraph *PG* to construct both the local scope of intentions *LSI* and local scope of capabilities *LSC* for the actor. We define a sub-procedure *ADD-To-LS* that takes an actor object *ar* and a action node *act* in the PlanGraph to add the action to the local scope *LSI* or *LSC* based on whether the corresponding relation exists. The *ADD-TO-LS* procedure then calls itself on all the children nodes to traverse to the deeper levels in the PlanGraph. In this way, the process to construct the local scopes *BUILD-LS* is composed of the initialization of *LSI* or *LSC*, and then a recursive call of *ADD-TO-LS*, beginning on the root node of the PlanGraph.

```
1: procedure BUILD-LS(ar, PG)
2:     LSI ← [ ], LSC ← [ ]
3:     root ← PG.root
4:     ADD-TO-LS(ar, root)                    ▷ start the recursion from root
5: end procedure
6: procedure ADD-TO-LS(ar, act)
7:     for all int in act.intentions do       ▷ check the current node
8:         if int.actor == ar then
9:             LSI.add(ar, act, int)
10:        end if
11:    end for
12:    for all cap in act.capabilities do
13:        if cap.actor == ar then
14:            LSC.add(ar, act, cap)
15:        end if
16:    end for
17:    for all par in act.parameters do       ▷ recursion on parameters
18:        for all subact in par.subacts do
19:            ADD-TO-LS(ar, subact)
20:        end for
21:    end for
```

```
22:     for all cond in act.conditions do                ▷ recursion on conditions
23:         for all subact in cond.subacts do
24:             ADD-TO-LS(ar, subact)
25:         end for
26:     end for
27:     for all subact in act.subacts do                 ▷ recursion on subacts
28:         ADD-TO-LS(ar, subact)
29:     end for
30: end procedure
```

If there are multiple actively PlanGraph, the local scopes can be generated by calling $BUILD\text{-}LS$ on each PlanGraph, and then merge the local scopes of intentions LSI and local scopes of capabilities LSC separately.

Show a result of the constructed local scope in the same concrete example of plangraph in the motivating scenario

5.2.4 Constructing dependency network

From the PlanGraph model, we can also dynamically construct the corresponding dependency network to represent how the actions are dependent on each other. Formally, a dependency network $DN = (V(DN), E(DN), \psi)$ is defined as a directed graph with the following characteristics:

1. $V(DN) = \{ACT \cup PARAM \cup COND\}$ is the set of all the nodes in the dependency network, all the *dependers* and *dependees* are action nodes ACT, and the *dependums* can be any parameter RES or condition nodes $COND$.

2. The set $E(DN)$ is a set of links between the nodes. Each link can be represented as a pair of nodes in $V(DN)$, i.e. $\psi : E(DN) \to V(DN) \times V(DN)$. The link can indicate a direct dependency between two action nodes, or a incoming link that connects a *depender* and *dependum*, or an outgoing link that connects a *dependum* and *dependee*.

The construction of dependency network can be also achieved by depth-first traversing through the PlanGraph model to search for all the action-action and action-resource relations. We define a procedure $BUILD\text{-}DN$ that constructs a dependency network from a PlanGraph PG. As the set of nodes $V(DN)$ can be calculated from the set of links $E(DN)$ by removing all the duplicate nodes in

$E(DN)$. The goal of the $BUILD\text{-}DN$ is to build $E(DN)$. Similarly, we define a sub-procedure $ADD\text{-}To\text{-}DN$ that takes an action node *act* in the PlanGraph to add dependencies that starting from the node *act*, i.e. the *act* as the *depender*.

```
1: procedure BUILD-DN(PG)
2:     E ← [ ]
3:     root ← PG.root
4:     ADD-TO-DN(root)                          ▷ start the recursion from root
5: end procedure
6: procedure ADD-TO-DN(act)
7:     for all subact in act.subacts do                    ▷ check for Sub.Act
8:         E.add(act, subact)
9:     end for
10:    if act.hasParent() then
11:        for all constr in act.parent.constraints do     ▷ check for Precedes
12:            if constr.next == act then
13:                E.add(act, constr.prev)
14:            end if
15:        end for
16:        for all par in act.parent.parameters do         ▷ check for parameters
17:            for all subact in par.subacts do
18:                E.add(act, par), E.add(par, subact)
19:            end for
20:        end for
21:        for all cond in act.parent.conditions do        ▷ check for conditions
22:            for all subact in cond.subacts do
23:                E.add(act, cond), E.add(cond, subact)
24:            end for
25:        end for
26:    end if
27:    for all par in act.parameters do                    ▷ recursion on parameters
28:        for all subact in par.subacts do
29:            ADD-TO-DN(subact)
30:        end for
31:    end for
32:    for all cond in act.conditions do                   ▷ recursion on conditions
33:        for all subact in cond.subacts do
34:            ADD-TO-DN(subact)
35:        end for
```

```
36:     end for
37:     for all subact in act.subacts do                    ▷ recursion on subacts
38:         ADD-TO-DN(subact)
39:     end for
40: end procedure
```

Show a result of the constructed dependency network in the same concrete example of plangraph in the motivating scenario should consider the state of the actions here or leave it for belief propagation? how to handle multiple PlanGraphs? shared resource, what else?

5.3 Representing Events

In Section 4.2.2.1, we discuss the difference between *'occurrence'*, *'awareness'*, and *'event'*, and define *'events'* to refer to the computerized entities that are used in an awareness system to represent knowledge about either real world *'occurrence'* or the results of *'awareness'* processes. Before delving into how events can be used by the computer system to update the knowledge representation of the field of work, or used by the human actors to develop awareness, we need to discuss how they are computationally represented and what types of events are defined in this study. In this section, we first describe the general representational structure of events, then discuss the major types of events, and how they are represented differently.

5.3.1 Structure of events

We adopt the similar approach with many existing event processing systems [67] to represent each event as a structured object consisting of a named set of attributes. Formally, an event e is a nonempty set of *attributes* $\{a_1, a_2, ..., a_n\}$, where each a_i a name/value pair (n_i, v_i) with name n_i and value v_i. It is assumed that names are unique, i.e., $i \neq j \Rightarrow n_i \neq n_j$, and that there exists a function that uniquely maps each n_i to a data type T_i that is the type of the corresponding value v_i.

The set of attributes for each event should help answer questions such as this: What occurrence or awareness aspect it refers to? When did it happen? Where did it happen? What other information is associated with its happening? The answers to these questions are usually depend on the type of event they are associated with. An *event type* is a generalization for a set of event objects that have

the same semantic intent and same structure [36], i.e. they share the same set of attributes, but may have different values. Each *event type* has a unique event type identifier. In this study we use simple descriptive text strings for these identifiers, for example the phrase *"LocationChanged"* identifies an event type that can describe any instance of an object's location change. Identifying the set of *event types* is an application specific task, as actors in different applications have different awareness needs and capabilities to detect events. We describe the major event types that are supported in this study in Section 5.3.2.

As arbitrary attributes can be included in each event type, we can distinguish between three kinds of attributes carried in each event, following the definitions in [36] (Figure 5.3):

1. The *header* consists of generic information about the event, such as the event type and occurrence time, etc. The name and meaning of these header attributes are not specific to a particular event type.

2. The *payload* contains a collection of attributes carrying the data that describes the actual occurrence. Unlike header attributes independent of the actual event type, the payload attributes are defined per event type.

3. An event can also contain free-format *open content* information that provides a mechanism that an event producer or the awareness system can use to enrich an event object with extra contextual information, such as human-readable explanation, multi-media content etc.

Header The header of an event contains the common attributes that are included in every event object. Unlike payload or open content that are optional, a header is required for every event representation. In general, the header of an event needs to include the following attributes:

1. *Event type.* This attributes stores the event type identifier that uniquely identifies the event type of this event.

2. *Event identifier.* This is a unique identifier for each individual event object.

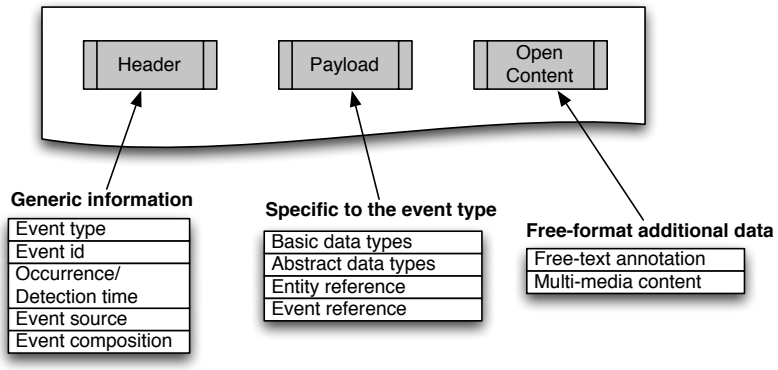


Figure 5.3. The structure of an event (adapted from [36] p.63)

3. *Occurrence/detection time.* The occurrence time attribute records the time at which the real world occurrence happens. In some cases, the event producer might not be able to determine the time when the event actually occurred. For example, if the producer examines the state of some external entity only at periodic intervals. In such cases, the detection time is used instead that records the time at which the event became known by the event producer.

4. *Event source.* This is the entity that originates this event. This can be either an external sensor, or a human actor in the collaborative system.

5. *Event composition.* This is a boolean attribute that denotes whether the specific event is a composite event or not. A composite event is one whose payload is made up of several different event instances.

Payload The attributes that make up the event payload are used to carry the data that describes the actual occurrence. The set of attributes included in each event is a variable that depends on the corresponding event type. There are several types of data that can be included as payload attributes:

1. *Basic data types.* The value in an payload attribute can simply be in the basic data types, such as string, numeric boolean, date/time etc.

2. *Abstract data types.* Attributes can also have abstract data types that are structures composed of other data types. For example, many events includes

a geographic attribute to records the whereabout of the represented real world occurrence. This attribute can be a point-based representation as a latitude/longitude pair, or more complicated as a route or a geographic area, which are in abstract data types.

3. *Entity reference.* Instead of records the information directly in an attribute, the event can also records information by pointing to entities represented in the field of work. For example, an *'ActionPerformed'* event may use the reference to the action node stored in the PlanGraph model to indicate which action is performed.

4. *Event reference.* Some events may contain references to other events. For example, a composite event may use the event references to records the primitive events that it is composed of.

Open content The open content of an event can include any attributes an event producer or the awareness system can use to provide additional contextual information about the event. For example, it is used in the awareness externalization process to allow the actors to provide the human-readable explanation of their interpretations.

5.3.2 Event types

As we argue in Section 4.2.2.1 that *events* can be used to represent both description of real world *occurrences* and externalization of human actors' internal *awareness* knowledge, a fundamental distinction should be made between these two categories of events, we call the former *external events*, and the latter internal events. The distinction between external events and internal events are important, because (1) they require different representational structures, i.e. the payloads of events have different set of attributes; (2) they are consumed differently by the system when updating the knowledge representation of the field of work; (3) and they are treated differently in human users' awareness processes.

Another distinction to make is the difference between *primitive events* and *composite events*. Composite events prevent the users from being overwhelmed by

a large number of primitive event by providing them with a higher-level abstraction [67]. Generally, a composite event is made up of several other events (either primitive or composite), according to a specification of relations between them.

In the following, we first describe the major primitive event types in both external and internal categories, and then discuss the composite events as a special event type with its own payload structure.

5.3.2.1 External events

As we conceptualize a collaborative environment as consisting of a variety of entities and their relations, the external events can be defined to indicate any kinds of changes on either these entities or their relations.

Events on entities We define each external event on an individual entity as a semantic function that changes the entity's property or state. We follow the general event ontology proposed in [61] to define the following categories of external events on individual entities:

1. *State Change.* An event is a state change event if the occurrence yields a change of states on an entity. All the possible states of an entity are usually can be expressed as a discrete state machine, and each state transition indicates a possible state change event. For example, Figure 5.4 shows the transition diagram with all the possible execution states of an action. Each valid transition in the diagram can be considered as a state change event.

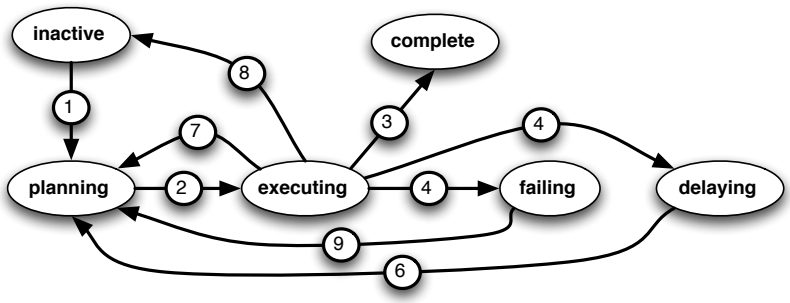


Figure 5.4. Execution state transition of an action

2. *Existential Change over Time.* An event indicates an existential change over time if its occurrence changes the existence of an entity in temporal order, e.g. an entity did not exist in the past but it exists now.

3. *Existential Change over Space.* An event indicates an existential change over space if its occurrence changes the existence of an entity depending on its movement through space, e.g. an entity existed at location A, but now exists at location B.

4. *Value Comparison.* Another common class of external events are comparison events to indicate changes on attribute values of an entity. They can used to indicate whether an attribute value is equal, unequal, greater than, or less than a fixed threshold, or the same attribute value in the past.

Events on relations The events on relations are used to indicate whether some relations between entities hold. For example, an event with the type *'ResourceAssigned'* indicates an assignment relation between a resource and an action starts to hold, i.e. the resource is now assigned for performing the action. Therefore, the types of external events on relations depend on the possible types of relations that can be identified in the domain.

Generally, the basic relations between entities in the real world can be divided into three categories: spatial, temporal, and conceptual [110].

1. The spatial relations link the entities through their spatial positions. The basic types of spatial relations have been well studies in the literature on geographic information systems, include binary topological [30], directional [40], and distance relations [54]. Topological relations is a particular subset of geometric relations that are preserved under topological transformations such as translation, rotation, and scaling. Some examples are relations indicating whether one entity disjoints, meets, overlaps, or contains another entity. Directional relations indicate the relative direction between two entities, such as one is at north of the other. Distance relations link entities based on their proximity in the space, such as one is within a certain range of another.

2. The temporal relations link the entities based on their temporal positions. The basic types of temporal relations are considered in the literature on

temporal reasoning [3], including binary topological, ordering, and distance relations [4].

3. The conceptual relations are an umbrella term that cover all the different types of organizational, structural, or social relations between entities in a particular collaborative activity.

From the basic types of relations, more complex types of relations can be built, such as density (clustering, dispersion), arrangement (e.g. sequence in time or alignment in space) and spatial-temporal relations. The latter are composed of spatial and temporal relations and represent changes of spatial relations over time: approaching or going away, entering or exiting, following, keeping distance, concentrating or dissipating and so on.

Figure 5.5 shows an upper level typology of the external events that can be defined on entities and relations in a collaborative environment.

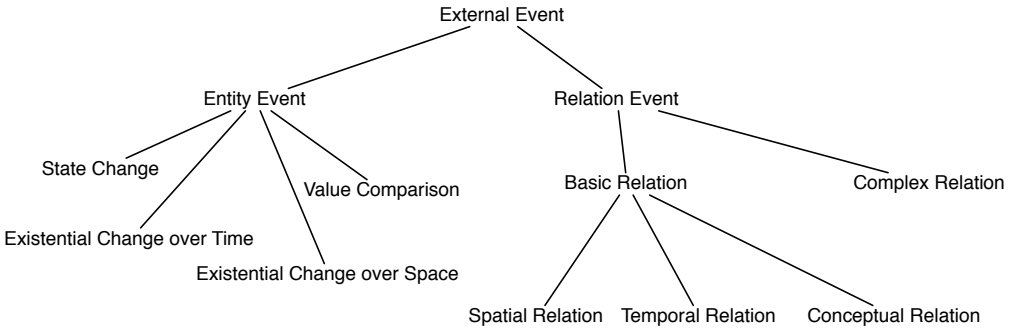


Figure 5.5. An upper level typology of external events

Relevance to the field of work Although the number of external events that could be possibly identified is infinite, not all of them are relevant to the human actors' working context, i.e. the field of work. As a result, our goal of identifying external events focuses on finding a subset of event types that are relevant in an application domain. Existing studies to identify such a subset of event types are usually done in two ways [77]: from the 'supply' side to identify the set of event types that can be recognized by available sensors, and from the 'demand' side to define event types based on the nature of the activities supported by the event-based systems. As we are more concerned about the human actors' awareness needs

in performing their collaborative activities, we follow the 'demand'-side approach and identify the major event types based on how they could possibly contribute to the understanding of the field of work in a collaborative activity.

In general, we believe that the relevance of external events to the field of work can be analyzed based on the different functional roles they can play in updating the knowledge about the field of work. If the occurrence of an external event can imply some change in the field of work, it should be treated as relevant. Based on our conceptualization of the field of work, we can identify the following function roles for external events:

1. *Direct change to entities in the field of work.* The external events can indicate changes on individual entities that are modeled in the field of work, i.e. the property or state change on the resources, actors, or actions. For example, a state change event can be used to indicate the change of execution state for an action in the field of work. Location change events can be used to describe an actor's movement in the field of work.

2. *Direct change to relations in the field of work.* Within the different types of relation events that can be possibly identified, some of them are directly related to the various relations between resource, actors, and actions as we described in Section 5.1.1.2. For example, an external event type indicating the constitution relation between two entities can be used to describe the *Sub.Act* relation between two actions, i.e. one action is a subsidiary action to perform another one. The assignment relation event types can be used to describe relations between an actions and a resource, i.e. the resource has been assigned to the performance of the action.

3. *Action motivation.* Aside from the two cases that external events can be directly linked to the entities and relations in the field of work, the external events can also impact the field of work by motivating the actions that need to be performed. For example, an external event indicating that the fire alarm is ringing will activate my goal to escape from my office. In this case, the fire alarm is not directly linked to any entities in my current field of work, rather it motivates me to perform a new action.

4. *Indication of action performance.* In some cases, the external events can also provide some evidence implying the effects of action performance. For example, instead of a state change event directly showing the action to delivery a resource to an actor has been completed, it could be implicitly inferred from a spatial relation event that indicates the resource is now located at the actor's location. Similarly, an external event indicating the occurrence of a traffic blocking between the resource's current location and the actor's implies the delivery action is delayed.

5.3.2.2 Internal events

The internal events are used to describe the results of human actors' awareness processes. Unlike the external events can describe any entities and relations in the real world, they describe the changes of human actors' internal mental states in the field of work. The internal events are usually derived from the external events to indicate the human actors' interpretation on these external events. In this study, we identify two types of internal events: *intention* events and *belief* events.

An *intention* event indicates a human actor's adoption of certain intention towards some action in the field of work. As we described in Section 5.1.1.2, an actor's intention to an action can be of different kinds, i.e. $Pot.Int$, $Int.Th$, or $Int.To$, with different levels of commitment. Figure 5.6 shows the basic structure of an intention event. The payload of an intention event include three required attributes: an intention type indicating whether it's a potential intention, intention-that, or intention-to, a reference to the actor entity who has this intention, and a reference to the action that is intended to. An optional free-text attribute is included in the open content part of an intention event, where the human actor can provide the rationale for adopting the corresponding intention.

An *belief* event describes a belief of a human actor on some occurrence in the field of work. For example, it can be used to indicate an actor's belief that an action has been successfully performed, or his/her belief that he/she has the capability to perform an action. As belief events usually refers to some occurrence in the field of work, we represent belief events by embedding an external event representing the occurrence into its payload (Figure 5.7). However, unlike the standalone external event that indicates changes already happened, the occurrence described in a belief

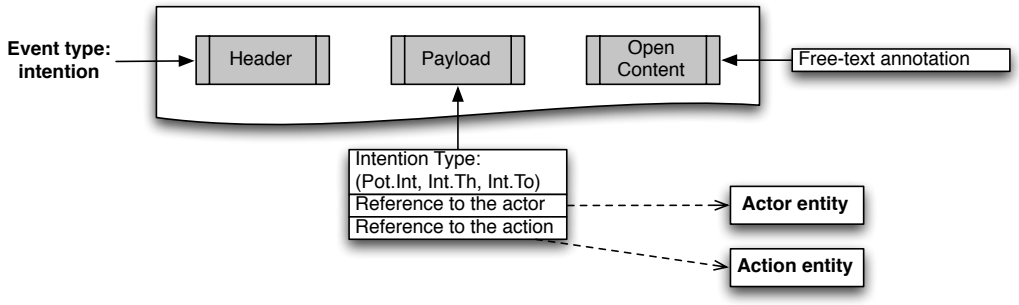


Figure 5.6. Structure of an intention event

event can be something that will happen in the future. These belief events represent the results of the human actor's projection process, i.e. what he/she expects to happen in the future. There are two attributes in a belief event's open content part: the free-text explanation of this belief, and a confidence level to describe how confident he/she is about this belief.

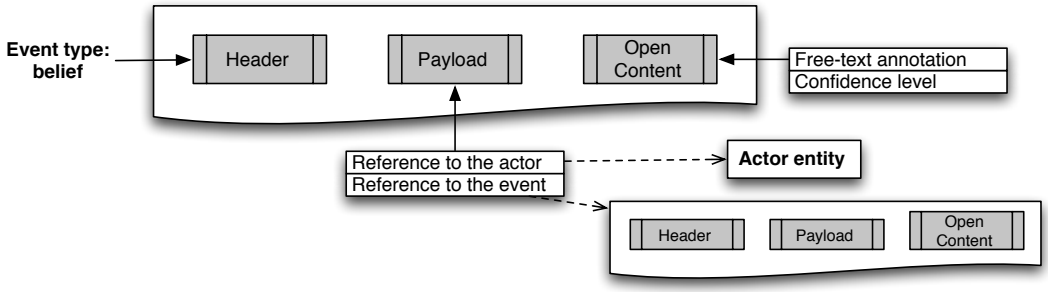


Figure 5.7. Structure of a belief event

5.3.2.3 Composite events

Composite events are a special type of events that can consist of several other events. The subsidiary events can be external or internal. Besides the list of subsidiary events, a composite event needs to also describe how these sub-events are combined together. In a simplest case, a composite event can occur only when all the sub-events occur. Moreover, a composite event can occur when some of the sub-events occur, but some do not. Or it occurs when any of the sub-events occur. A more complicated composite event language can be found in [67] that in-

cludes different relations between the sub-events, such as negation, concatenation, sequence, iteration etc. To describe the composition pattern, a specific payload attribute needs to be included in a composite event's representation (Figure 5.8).

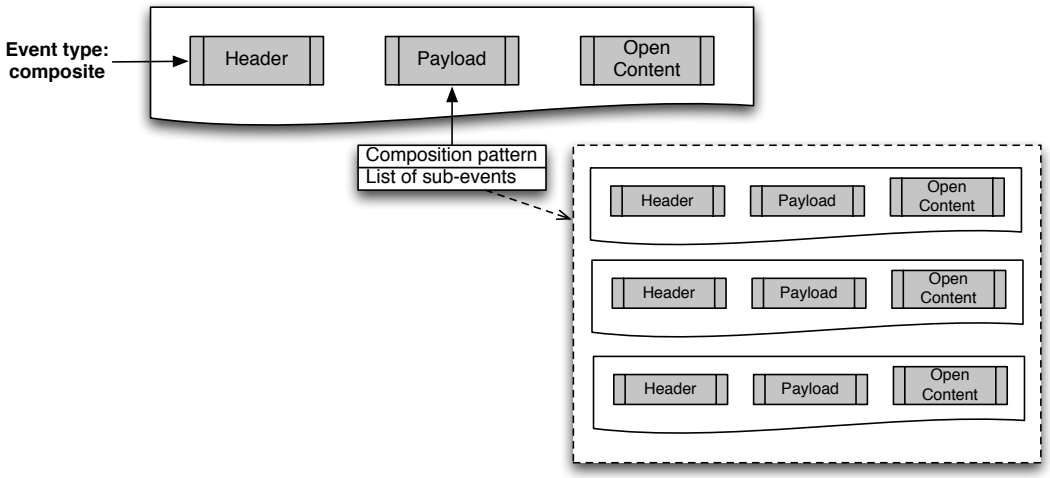


Figure 5.8. Structure of a composite event

5.4 The knowledge updating process

Each time when a new event is input into the system, it triggers the knowledge updating process. The knowledge updating process performs two important tasks: (1) it decides on how the input event influences the state of the field of work and updates the correspondent in the PlanGraph model; (2) it augments the event representation by establishing the links between the event and the corresponding entities in the PlanGraph, and records the development of the event based on the system's reasoning.

Updating the PlanGraph In general, the knowledge updating is a four-step process of *association*, *assessment*, *elaboration*, and *propagation*, through which the knowledge representation of the field of work, i.e. the PlanGraph model is updated (Figure 5.9).

1. *Association.* The knowledge updating starts with the association of an event with the PlanGraph model. In this step, the system searches the PlanGraph

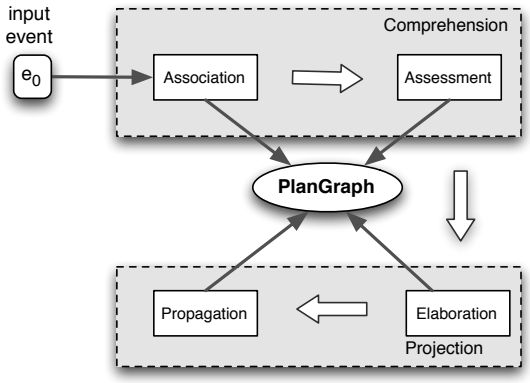


Figure 5.9. The knowledge updating process

for an appropriate match between the input event and the entities or relations in the field of work. If a match is found, the system uses the information stored in the event to update the corresponding entities or relations in the PlanGraph.

2. *Assessment.* The second step is to assess how the event can contribute to new changes in the collaborative activity. It may trigger new actions that need to be added to the PlanGraph, or change the current state of existing actions.

3. *Elaboration.* Based on the assessment of new changes, the elaboration step is to reason about the system's expectation on any new actions that need to be added to the PlanGraph model, or what actors will be potentially involved in these new actions. This is usually performed with domain specific knowledge, such as recipes of performing an action, and the specification of role responsibility.

4. *Propagation.* The propagation step focuses on evaluating how the current change can be possibly propagated other actions in the PlanGraph model because of the dependencies among them.

The four steps of knowledge updating process share some commonality with the human actor's awareness development processes. The *association* and *assessment* steps are very similar to the *comprehension* process, where the actor comprehend

or understand the relevance of awareness information in relation to its tasks and goals. The *elaboration* and *propagation* can be considered as the projection of the states in the near future, as the actor forecasts likely future states in the situation. In this way, we can think of the knowledge updating process as the computer system's awareness development process. The only difference is that, as the human actor usually only needs to be aware of things in his/her local scope of work, the computer system aims to possesses the knowledge of the whole field of work through the knowledge updating.

One thing to note is that not every event will be processed through all the four steps. Some external event may not directly link to any entity in the field of work, or the system does not have the complete knowledge to assess its implication on the field of work. In such case, the event will be passed by to the human actors for interpretation. The result of human actor's interpretation as a new internal event then starts a new round of knowledge updating, during which the system's knowledge is updated.

Updating the event representation As we emphasize in the beginning, while the PlanGraph model of the field of work is updated by the new event, the event itself is also developed in the system's reasoning processes. For example, in the *assessment* step, an external event *'TrafficBlocked'* causes the system to believe that the action to delivery a resource to an actor cannot be achieved, i.e. One one hand, this causes the system to modify the state of the delivery action in the PlanGraph model. Meanwhile, the system generates a new internal event describing the system's belief about the state change on the delivery action. Later, in the *propagation* step, the system may generate a new internal event to indicate that because of the state change on the delivery action, the actor's action that is waiting for the resource delivery will also be impacted. In this way, the original event is derived into a chain of events as the knowledge updating process proceeds.

To record the development of an event in the knowledge updating process, we define an *event chain* EC as an ordered sequence of events: $EC = (e_0, e_1, e_2, ...)$. In the beginning of the knowledge updating process, there may be only one event e_0, i.e the original external event in the event chain EC. As the knowledge updating proceeds, more events are added to the chain. In the assessment step, the system

may generate *derived events* indicating the system's belief on how the field of work has been changed due to the original events. In the propagation step, the system predicates the future state changes, and attaches more *anticipatory events* to the chain. Figure 5.10 shows the knowledge updating process with the development of the event chain.

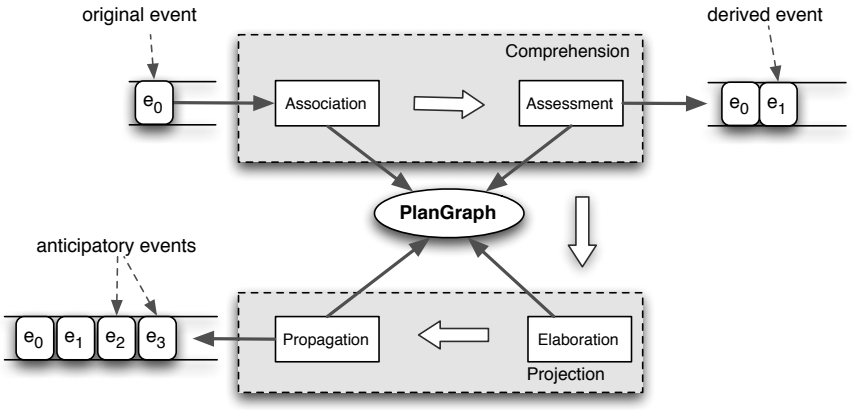


Figure 5.10. The development of an event chain

Within the structure of our event representation (Section 5.3.1), the *event chain* is achieved by three additional attributes in the *open content* section of an event:

1. A text string (*label*) indicates the functional role of the event in the event chain, i.e. whether the event is an *original* event, a *derived* event, or an *anticipatory* event.

2. An event reference (*prevEvent*) points to its preceding event in the event chain. It can be optional when the current event is an *original* event, but required for *derived* and *anticipatory* events.

3. An event reference (*nextEvent*) points to its succeeding event in the event chain. It can be optional if the current event is the end of an event chain.

Figure 5.11 shows how the information about the event chain is included in an event's representation.

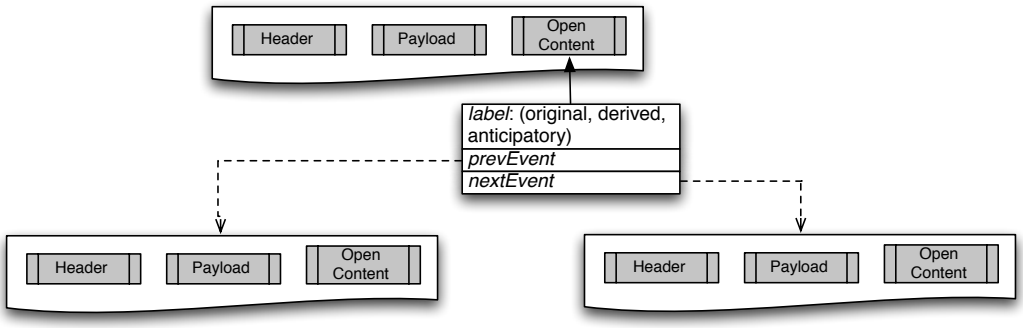


Figure 5.11. Event structure in an event chain

5.4.1 Association

The knowledge updating starts with establishing the association between the input event and the current PlanGraph model representing the field of work, and use the event information to update the PlanGraph model. The association starts with the recognition of event type for each input event, as the different event types are treated differently.

If the event is an *external event* on entities, we consider the following two cases:

1. If the event is indicating an existential change over time, the event is directly passed through to the assessment step, as the entity linked to this event did not exist in the past.

2. Otherwise, we search all the PlanGraph nodes to find any match with the entity described in the event based on their unique identifiers. If a match is found, we update the attribute information attached with the PlanGraph node based on the event type. If it is a state change event, we change the state of th node. If it is an existential change over space, we update the location information of the node. If it is a value comparison event, we update the corresponding attribute value. After updating the node, we add an entity reference in the event object, so that it can direct link to the matched PlanGraph node for later use.

If the event is an *external event* on relations, we check for the following cases:

1. If the event indicates a resource assignment relation, i.e. a resource res_1 is assigned to the performance of an action act_1. we search all the action nodes

in the PlanGraph to find any match with the action entity described in the event act_1. If a match is found, we search for the parameters of act_1 in the PlanGraph to find any of them has the same resource type as res_1 and then assign the values of res_1 to the parameter. After that, we add an entity reference pointing to the parameter node in the event's payload.

2. If the event indicating a structural relation, e.g. an action act_2 is a subsidiary action of another action act_1. we search all the action nodes in the PlanGraph to find any match with the action entity act_1 described in the event. If a match is found, we create a new node with the information described in act_2, and then add an entity reference pointing to this new node into the event's payload. This can be applied to other decomposition relations, such as an action is a subsidiary action to achieve a condition, or a new parameter needs to be added to an existing action.

If the event is an *internal event*, we consider the following cases:

1. If the event is an *intention event*, we search all the action nodes in the PlanGraph to find any match with the action entity described in the event. If a match is found, we search for the *Intentions* associated with the action node to find any existing actor has the same identifier as the actor entity described in the event. If so, we update the intention type based on the intention event. Otherwise, we add the actor and the corresponding intention into the the *Intentions* associated with the action node. After that, we add an entity reference pointing to the action node in the event's payload.

2. If the event is a *belief event* about an actor's capability to perform an action, we search all the action nodes in the PlanGraph to find any match with the action entity described in the event. If a match is found, we search for the *Capabilities* associated with the action node to find any existing actor has the same identifier as the actor entity described in the event. If so, we update the capability type based on the belief event. Otherwise, we add the actor and the corresponding intention into the the *Capabilities* associated with the action node. After that, we add an entity reference pointing to the action node in the event's payload.

3. If the event is other *belief events*, we apply the association rules directly on the subsidiary event describing the content of the belief.

In sum, two results are achieved if an association between the event and the PlanGraph model is found in this step. First, the corresponding PlanGraph entity or relation is updated based on the new information carried by the event. Second, the event is enriched with a direct link to the corresponding PlanGraph node, that is the context of origin of this event is identified.

However, not all the events are directly associated with the PlanGraph model. As we discussed in Section 5.3.2.1, some external events may describe occurrence on the entities that are not currently in the PlanGraph, but implicitly impact the field of work by motivating new actions or implying the effects of action performance. These events will be passed through to the assessment step for further analysis.

5.4.2 Assessment

In the assessment step, each event is evaluated to check how it can lead to changes towards the action performance, such as the execution state change of an action, or a condition. The changes towards action performance can be explicitly expressed in the event, and is directly associated with the action node during the association step. For example, it can be an internal event from a human actor indicating the belief that the resource delivery action has been successfully performed. However, in the assessment step, the system is more interested in the events that impact the action performance implicitly, where inference becomes necessary. For example, an external event indicating that a resource is now located at an actor's position can implicitly indicate the successful performance of the resource delivery action. In this case, a new event showing the state change of the resource delivery action will be derived and added to the event chain along with the original event.

The knowledge stored in the PlanGraph allows the system to perform some routine assessment tasks that are universal across different application domains. Some of the basic inference tasks can be described as follow:

1. *Goal conditions on action nodes.* If an event describes changes on entities or relations that are included in an action's goal condition, the system can evaluate the action's goal condition. If the goal condition becomes holding

because of this event, we derive a new state change event on this action, and push it into the event chain.

2. *Condition nodes*. If an event describes changes on entities or relations that are included in a condition node, the system can evaluate the condition node. If the condition becomes holding or no longer holding because of this event, we derive a new state change event on this condition, and push it into the event chain.

3. *Parameter nodes*. If an event describes changes on an resource that is assigned to a parameter node, the system can evaluate the parameter's subsidiary conditions. If a condition becomes holding or no longer holding because of this event because of this event, we derive a new state change event on this parameter, and push it into the event chain.

The second type of tasks that the system can perform during the assessment process is to check whether the event can activate new action that needs to be added to the field of work. This type of events is usually called *triggering events*, as they are often not directly associated with any current nodes in the PlanGraph, but will trigger some new action to be performed. For example, every time a new victim is found in an emergency response operation will trigger a new rescue action to be performed. In this case, the initial event about the discovery of a new victim cannot be associated with any existing actions, but asks for a new action to be performed. The assessment of action activation requires a set of pre-defined domain-specific activation rules, so that every event can be searched through the activation rules to find whether it satisfies any of the conditions. If so, a new action is added to the field of work as a new PlanGraph, and the new derived event is generated to indicate the activation of the new action.

Furthermore, the assessment step can involve more sophisticated inference techniques, such as spatio-temporal reasoning [10], pattern recognition [120], or case-based reasoning [59], to enhance the system's reasoning capabilities. However, these reasoning techniques often require a great number of domain knowledge, and lack flexibility to handle unexpected events. As we discussed in Section 4.1.2, one of the design principles of our approach is to leverage the different capabilities of computers and humans. As a result, in the assessment step, we design the system

to focus on the more reliable low-level routine inferences that can be directly performed in the PlanGraph model, and leave the complex, higher level assessment to the human actors. Hence, some events may not be considered as contributing to the collaborative activity by the system in the assessment process. Rather, they are later interpreted by human actors and send back to the system as internal events, which will then be used to update the system knowledge.

5.4.3 Elaboration

The main goal in elaboration step is to advance the collaborative activity from the system side based on the change from the new event. After the assessment process, the context of the activity is changed, and the system attempts to elaborate the PlanGraph to accommodate the changes.

Based on the specification of SharedPlan theory [47], the system can elaborate the current PlanGraph in several ways.

1. *Recipe selection.* The system can contribute to the collaborative activity by retrieving a recipe for a new action from the knowledge base, i.e. by providing a default way to perform this new action.

2. *Parameter binding.* If any of the parameters is unbound to any values, the system will search the knowledge base to find any action that can be performed to identify the value for the parameter.

3. *Condition satisfaction.* If any of the pre-conditions is not holding, the system will search the knowledge base to find any action that can be performed to satisfy the condition.

4. *Actor allocation.* If any of the actions has not been committed by any actors, the system search for the actors who might be potentially intended to or capable of performing the action.

The elaboration step is not performed for every event, rather it is only triggered by a subset of events that are related to the development of the collaborative activity.

1. Events on structural relations. Whenever an event indicates that a new action is a subsidiary action to another action, a way to identify a parameter, or to achieve a condition, the new action will be added to the PlanGraph in the association step, and needs to be elaborated.

2. Events on goal activation. The elaboration needs to be performed whenever a new action has been added to the field of work because of some triggering event during the assessment step.

3. Events leading to condition violation. Whenever an event leads to the fact that some condition is no longer holding in the assessment step, the elaboration needs to be performed on the condition node to identify any action that can be performed to satisfy it.

The elaboration process is achieved with the support of two types of pre-defined knowledge: (1) the recipe knowledge about how to derive an action into a sequence of parameters, pre-conditions, and subsidiary actions, how to identify a unbound parameter, or how to satisfy a condition, etc.; (2) the knowledge about actor roles and their corresponding responsibilities and capabilities. The former allows the system to extend the field of work by adding new action, parameter, or condition nodes into the PlanGraph model, and the knowledge about actor roles allow the system to reason about who are likely to be interested in these newly added entities, or who have the capability to work on these new entities, so that the system can extend the actors' local scopes to these newly added entities.

The elaboration process is predictive as it reflects the system's prediction on how the collaborative activity will be advanced based on the new event occurrence. The system provides a default plan of performing an action, and identifies the potential actors who might be interested in it. After the elaboration process, the PlanGraph not only reflects the current state of the collaborative activity, but also shows the potential next steps based on the system's knowledge. However, the results of elaboration process never dictate how the human actors will eventually develop the action. The human actors may later generate new internal events to revise the plan generated by the system, or change their intention/capability levels towards the action in the PlanGraph.

5.4.4 Propagation

Propagation is another predictive process that the system can perform to predict future state changes in the field of work. It is triggered by the events that either directly indicate (in the association step) or imply (in the assessment step) state changes on the entities in the filed of work, and then reason about how these initial state changes can be propagated to other actions due to the multiple dependencies between them. A simple example could be be: if the execution state of an action act_1 is changed from *executing* to *failed*, then the parent action act_2 (i.e. $SubAct(act_1, act_2)$ holds) will likely to be impacted and may be also changed to *failed* if the plan is not changed.

The propagation is performed on the dependency network, which can be easily constructed from the PlanGraph model (Section 5.2.4). The dependency network abstracts away the detailed information on each action node and focuses on the dependency relationships among them, we can adopt more efficient network-based reasoning models to perform the propagation. In this study, we employ the Bayesian network to perform the propagation [74]. Bayesian networks are directed acyclic graphs in which the nodes represent multi-valued variables, and the arcs signify direct dependencies between the linked variables and the strength of these dependencies are quantified by conditional probabilities. The purpose of the Bayesian network is to give a belief in each possible value for each node after some evidence arrives.

In the context of our model, the nodes are the basic elements in a dependency network, i.e. dependers, dependums, and dependees, which represent the entities in the field of work, i.e. actions, resources, or conditions. The evidence fed into the network includes certain state changes on these entities. Thus, the purpose of the Bayesian network is to update the systems belief in the states for every other node after some state change occurs on a node.

To operationalize the Bayesian network, we need to follow the following steps: (1) construct the Bayesian network, (2) assign the conditional probabilities for each link, (3) and perform the belief updating when some events arrive.

Construction of Bayesian networks By following the algorithm in Section 5.2.4, we can construct the dependency network from the PlanGraph model, and

then the construction of Bayesian network is very straightforward. We translate each basic element in the dependency network into a node variable, and the different dependency relationships into corresponding links in the Bayesian network. The possible values for each node are determined based on the possible execution states for each type of node variables. At a given time, an action can be at one of the following states: *inactive*, *planning*, *executing*, *complete*, *failing*, and *delaying*. Each condition can be *open*, *waiting*, or *holding*. Each resource can be *unavailable*, *waiting*, or *available*. Figure 5.12 shows all the states for each type of node and the possible state transitions.

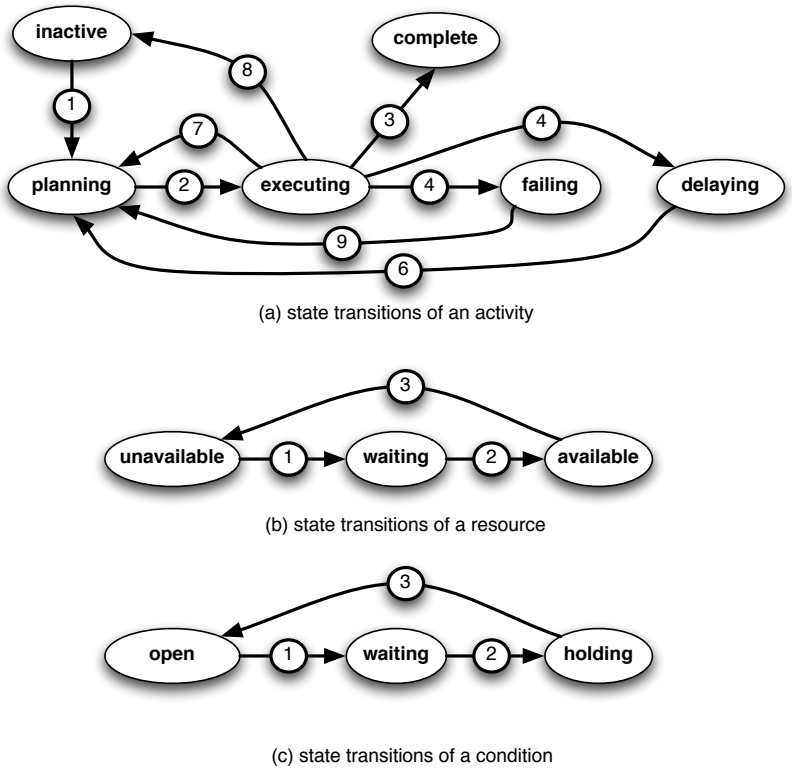


Figure 5.12. Types of node variables and their possible values in a Bayesian network

Assignment of conditional probabilities Before any propagation commences, we need to assign the conditional probabilities for each node that form the conditional probability matrices. An element of the conditional probability matrix looks like $P(x_i|u_{1j_1}, ..., u_{nj_n})$, and gives the probability of state i for node x conditioned

on the states of its parent nodes. For example, the conditional probability of an action node indicates the possibility of each state for this action, given the states of all the resources, conditions, actions that it depends on.

showing an example of the conditional probability table add an appendix showing the algorithm for the assignment

In our study, we develop a set of heuristic rules to assign the initial conditional probabilities for each type of nodes, by evaluating the criticality of each parent node, and the opportunities for re-planning. For example, if a critical resource is needed for every possible way to perform an action, the state of the resource as *failing* will definitely lead to the *failing* of the action. However, if there are other possible plans to perform the action that do not require this resource, the probability of the action being failing will be decreased.

The assignment of conditional probabilities provides the system with the default reasoning capability to propagate the state changes that can be later overwritten by the user's interpretation. When a user changes the belief in the state of an activity, the change will overwrite the systems initial belief through the belief updating. Because of this interactive nature, the purpose of the initial probability assignment is just to provide a good guess about the propagation from the system's perspective and does not have to be perfectly accurate.

Belief Updating We follow Pearls belief propagation algorithm [74] to perform belief updating whenever a state change occurs. In this approach, the belief in each value of a node variable is divided into two parts: the part emerges from its ancestors and the part emerges from its descendants, and the final belief is ascribed by multiplying the two parts. As a result, the belief updating is performed as a bidirectional propagation process.

1. Starting at the node where the initial state change occurs, the system calculates how the state change will change the beliefs on each parent node. If the change on a parent node is significant, i.e. above a given threshold, the parent node becomes active, and will be further propagated to its parent. This is called *causal* propagation since the updating is from a cause to an effect to indicate how a state change of the cause will lead to the change of the effect.

2. On the other hand, the belief updating can also calculated from an active node to their children. This is called *evidential* propagation as the reasoning flows from evidence to hypothesis.

In our approach, both the causal and evidential propagation will be performed. The causal propagation is used to provide the system's predicted state changes on the actions that are dependent on the action where initial state change occurs. The evidential propagation occurs whenever a user modifies the systems prediction on a given node and the system uses it as an evidence to trace back and revise the previous beliefs on the dependees.

The result of the Bayesian network-based propagation will be used to enrich the event chain with anticipatory events. Starting from the node that the initial event is linked to, the system use the previous probability distribution before the propagation and the current values to perform the Cartesian product on each node. Each value in the new two-dimension table indicates the probability of each possible state change. If a significant state change has been detected (by comparing with pre-defined threshold), it will be added to the event chain as a new anticipatory event. For example, Figure 5.13 shows the result of a propagation process on a parameter node. Before the propagation, the parameter had a high probability to be in the state of *waiting*, and after the propagation, the probability distribution changed. By calculating the Cartesian product, we can find the most significant state change on the node is from *waiting* to *delay*, with a confidence level of 0.83808. As a result, a new anticipatory event indicating that the state of this parameter has been changed from *waiting* to *delay* will be pushed into the event chain.

5.4.5 An Example

To demonstrate the knowledge updating process, we consider a simplified version of the victim rescue activity in the emergency response scenario to see how the PlanGraph model of the field of work is updated through several consecutive events, and meanwhile the events are enriched into event chains. The task involved in this example is quite simple. Whenever a victim is found, it needs to be rescued by sending to a medical station for treatment. We assume there is only one station and one medical professional (*med*) working at it. There might be several drivers

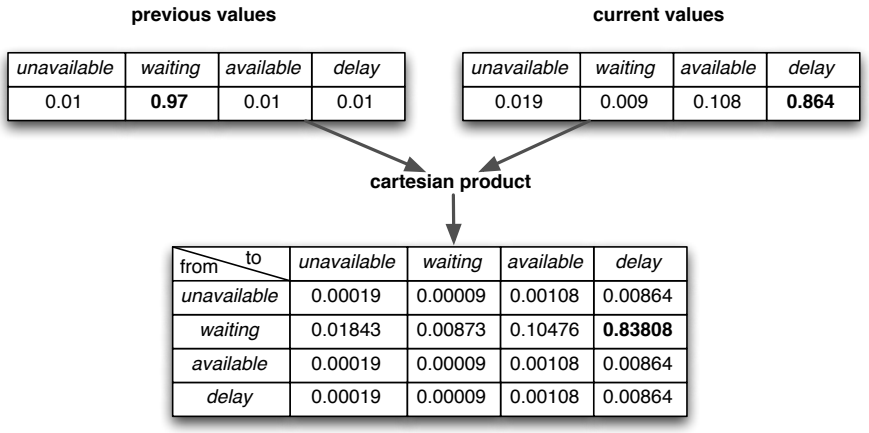


Figure 5.13. An example of calculating state-change probabilities

$(dr_1, dr_2, ...)$ with rescue vehicles that can deliver the victim to the station.

Event 1: *'NewVictim'* The first event sent to the system is an external event reporting that a new victim has been found. The event has an event type *'NewVictim'* and has several key attributes (e.g. id, occurrence time, location, required delivery time). Figure 5.14 shows the knowledge updating process performed on this event.

1. When the event is first sent to the system, the system attempts to associate with any existing entities in the PlanGraph model. Because this is the start of the activity, no association can be found.

2. Then the event is further processed in the assessment step, where the system checks whether the event can lead to changes towards the action performance or trigger new action. By checking the association rules stored in the knowledge base, the system finds that every *'NewVictim'* will activate the goal to rescue it. Following this activation rule, the initial PlanGraph is generated with just one action node (*'rescue'*) representing this new action to rescue the victim.

3. After the assessment, the system finds that a new action has been added to the field of work, which triggers the elaboration process. The system searches the knowledge base to find a recipe for the *'rescue'* action, and extend the

PlanGraph model with the parameters, conditions, and sub-actions. In this example, there is one parameter that is the *'victim'* who needs to be rescued, one condition (*'victimAtStation'*) that is the victim needs to be located at the medication station, and then the sub-action (*'medicalTreat'*) to perform the medical treatment on the victim. As these subsidiary entities are also new to the PlanGraph, the elaboration continues on each of them. During the elaboration of the parameter, the system assigns it with the value from the input event. The condition is elaborated with a new action (*'transport'*) to achieve it, which is further elaborated into subsidiary parameter and actions (*'vehicle'*, *'pickup'*, *'deliver'*). During the elaboration of action *'medicalTreat'* and action *'transport'*, the system also looks for the potential actors who might be interested in these actions. Based on the actor *med*'s role as a medical professional, the system believes that *med* has the potential intention (*pot.int*) and is able (*able*) to perform the *'medicalTreat'* action. Similarly, the system believes all the drivers $dr_1, dr_2, ...$ have the potential intention (*pot.int*) and are able (*able*) to perform the *'transport'* action.

4. Because the event does not indicate any state change on current actions, the propagation is skipped.

As we can see in Figure 5.14, after the knowledge updating process, the PlanGraph is updated to reflect the system's prediction on how the activity will be advanced. At the same time, the initial event is augmented with a new attribute pointing directly to the parameter node *'victim'* in the PlanGraph.

Event 2: *'IntentionChange'* The second event is an internal event that is generated by the driver dr_1. After he interprets the first *'NewVictim'* event, he decides that he has the intention to perform the *'transport'* action to deliver this new victim to the medical station. As a result, he generates this *'Intention'* event to update his intention on the *'transport'* action from *pot.int* to *int.to*. This *'IntentionChange'* event has several key attributes, including the actor's id (dr_1), the action he intends to perform (*'transport'*), and the intention type (*int.to*).

As the system receives this event, the system first attempts to associate it with any existing entities in the PlanGraph model. Because this is an intention event,

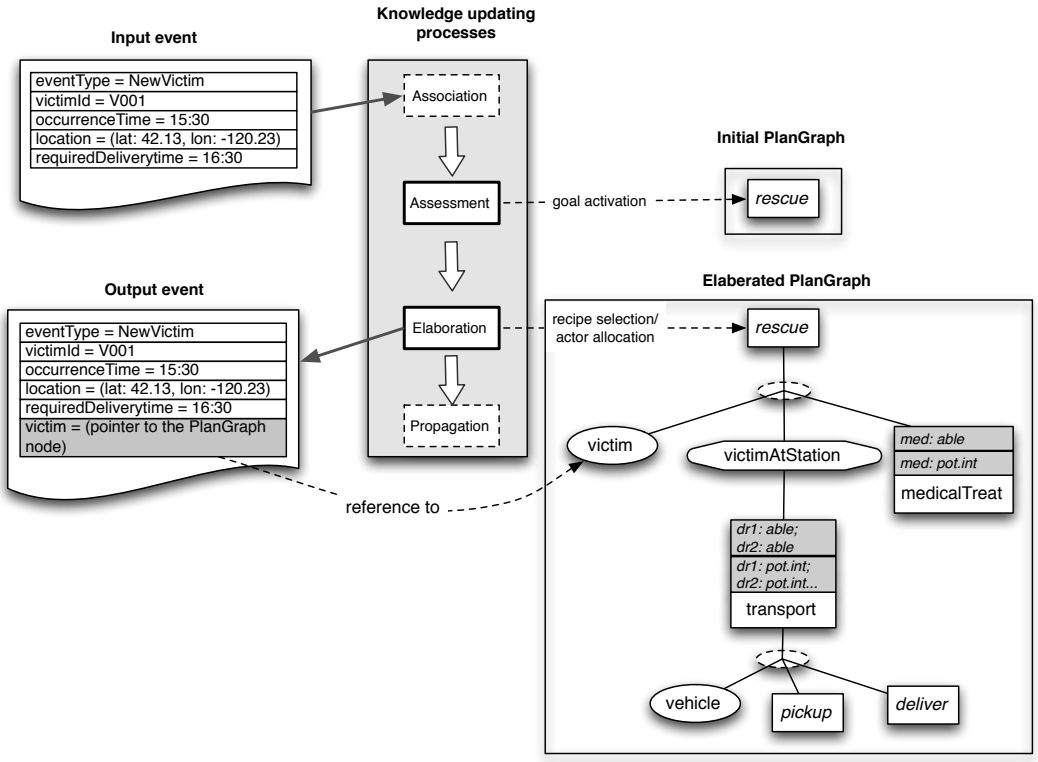


Figure 5.14. The knowledge updating example (*Event 1*)

the system traverses through the PlanGraph to search for any match between the action (*'transport'*) mentioned in the event and the action nodes in the PlanGraph. When the system is able to find such a match, the system uses the information in the event to update dr_1's intention level on action *'transport'*. Meanwhile, the corresponding attributes of the actor and the action in the event are updated with direct pointers to the nodes in the PlanGraph.

The knowledge updating process then proceeds to the following steps, but none of them leads to further changes in both the PlanGraph and the event itself. Figure 5.15 shows the whole process performed on the second event.

Event 3: *'FuelLevelLow'* The third event happens after driver dr_1 starts the action *'transport'* and is on the way to pick up the victim. It is an external event indicating that the fuel level on driver dr_1's vehicle is running extremely low. The event has an event type *'FuelLevelLow'* and carries information about the driver, the vehicle, and the current fuel level. Figure 5.16 shows the knowledge updating

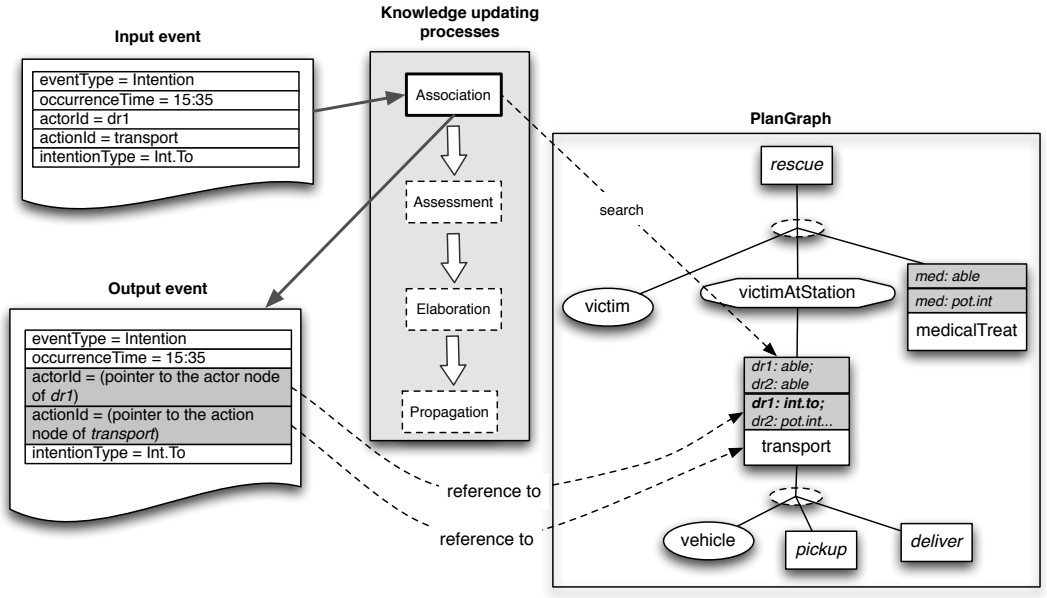


Figure 5.15. The knowledge updating example (*Event 2*)

process performed on this event.

1. When the event is first sent to the system, the system attempts to associate with any existing entities in the PlanGraph model. Because this is an external event indicating an attribute change on an entity, the system traverses through the PlanGraph to search for any match between the entity (*'vehicle'*) mentioned in the event and the nodes in the PlanGraph. When the system is able to find such a match, the system uses the information in the event to update the value attached to parameter (*'vehicle'*).

2. Then the event is further processed in the assessment step, where the system checks whether the event can lead to changes towards the action performance. By checking the conditions attached with the parameter (*'vehicle'*), the system finds that because of the low fuel level on the vehicle, the parameter becomes unavailable to perform the *'transport'* action. As a result, a new state change event *'ExecStatChange'* on the execution state of the parameter is derived, and added to the output event chain.

3. As no new action nodes are added to the PlanGraph, the elaboration process is skipped.

4. Because a state change event has been derived in the assessment step, the system attempts to predict future state changes in the field of work in the propagation step. A Bayesian network is constructed based on the current PlanGraph model and used to reason how likely the other entities in the PlanGraph will be impacted by the initial state change. After the Bayesian network-based reasoning, the system finds that the dr_1's action to pick up the victim (*'pickup'*) is likely to change its state from *executing* to *delaying*, and a new anticipatory event describing this state change on the execution state of the *'pickup'* action is generated, and added to the output event chain.

As we can see in Figure 5.16, after the knowledge updating process, the initial event is augmented into an event chain with three events: the original external event *'FuelLevelLow'*, the execution state change event on the parameter *'vehicle'* derived in the assessment step, and the anticipatory event predicting the execution state change event on the action *'pickup'* in the propagation step. This example shows the idea that not only the knowledge representation of the field of work, i.e. PlanGraph model is updated because of the event, but also the PlanGraph model can enrich the original event with system generated knowledge that is used to support the human actor's awareness processes in following chapter.

5.5 Discussion

In this chapter, we first focus on the knowledge representation of the field of work. We employ the PlanGraph model to represent the basic entities and relations in the field of work, and then use this model to derive the knowledge about local scopes of work and dependencies. The PlanGraph theory exhibits some important characteristics that make it suitable to satisfy the three requirements for representing the field of work described in Section 4.2.1.

1. The PlanGraph models collaborative activities as hierarchically structured subsidiary actions. The basic components of actions, parameters, conditions, and contraints can be used to capture all the basic activity entities and the relations between them can be used to derive the various dependency relationships.

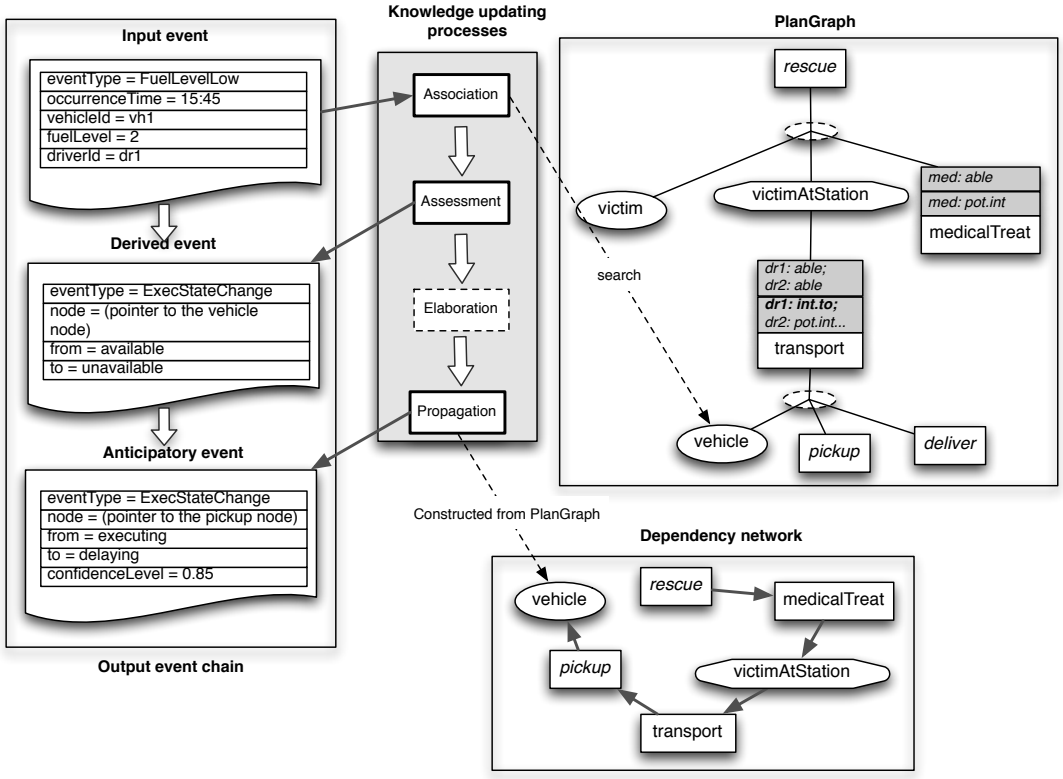


Figure 5.16. The knowledge updating example (*Event 3*)

2. The PlanGraph model also encodes the mental attitudes requirements of human actors in the actions, including different types of intentions and beliefs on their capabilities. This knowledge provides the basis to identify local scopes of participants.

3. A critical point made in the PlanGraph model is the emphasis on the development of collaborative activities. With the development of the activity, the PlanGraph model needs to be updated accordingly, so that it always models the current state of the field of work.

4. The PlanGraph model provides a set of reasoning capabilities that can be operationalized to support computer reasoning.

Following that, we discuss the representation of events and identify the major events supported in this study. The central idea of our approach is the interaction between these two knowledge components, which has been described in the

knowledge updating process. On one hand, the various events are consumed by the computer system to update its PlanGraph-based representation of the field of work. On the other hand, the PlanGraph model enrich the events with more meaningful contextual information, which is then used in next chapter to support the event-driven awareness processes.

Chapter 6

Promoting Event-Driven Awareness

In this chapter, we discuss our approach to promoting the event-driven awareness. More specifically, we address three major design components: (1) local space-based event notification mechanism, (2) a visualization framework for supporting event interpretation, and (3) interactive tools to support event propagation. The three components contribute to the different stages of the awareness development process. The event notification mechanism focuses on supporting the awareness perception by filtering out the events and controlling the notification styles. The visualization framework for event interpretation aims to support the development of higher level awareness, including both comprehension and projection. Supporting event propagation relates to the social processes to establish compatible awareness across multiple actors.

Each component is designed to follow the two design principles outlined in Chapter 4:

1. We emphasize how the computer's knowledge representation of the field of work and events are utilized in each component.

2. We attempt to strike a balance between the system's computational and reasoning capabilities to reduce human effort and providing interactive support to keep human actors in the loop and retaining control.

6.1 Event Notification Mechanism

As we discussed in Section 3.2.3, the event perception is usually supported by event notification mechanisms [65]. In this study, we distinguish *events* from *event notifications* as follow: while an event represents the information about a real-world occurrence or some aspect of actor's interpretations necessary to provide awareness, multiple notifications about that event can be directed to multiple receivers if it is relevant to them. Each event notification includes not only the content of the corresponding event, but also the actor who will be informed, as well as the notification style, i.e. the appropriate way to present the notification.

Event notifications are generated through event subscription and filtering mechanism. A subscription describes a set of notifications a user is interested in. The user register the interest in receiving certain kinds of notifications by submitting subscriptions to the computer system. The system filters incoming events based on the subscriptions and delivers those notifications that match one of the users subscriptions.

As we discussed in Section 3.2.3.1, several ways to specify interests and filter out events have been used in existing awareness systems, which offer different degrees of expressiveness and overhead on the users. Topic-based or type-based mechanisms are rather static and primitive, but can be implemented very efficiently and requires less effort for the user to manage the subscriptions. On the other hand, content-based mechanisms are highly expressive, but requires sophisticated protocols that have higher overhead to manage the subscriptions. This section provides the design of event notification mechanism in our approach. The major difference of our approach from existing studies is that we leverage the system's knowledge about each actor's local scopes to provide a two-tier notification mechanism that can combine the benefits of both topic-based and content-based mechanisms.

Our design of local scope-based event notification mechanism is based on the assumption that human actors in a collaborative activity are only interested in events that have impact on their individual working context, i.e. defined as their local scopes. Following this, our event notification mechanism is performed in two steps (Figure 6.1):

1. Given an event instance and an actor, the event is first filtered based on

whether it is associated with any entity in the actor's local scope of work.

2. If the event passes the first step, i.e. it is within the actor's local scope of work, the local scope-based subscriptions managed by the actor are applied to further filter the event and decide on the notification style for the event.

3. If the event does not pass the first step, it is sent to a standard content-based notification module to further decide on its relevance to the actors.

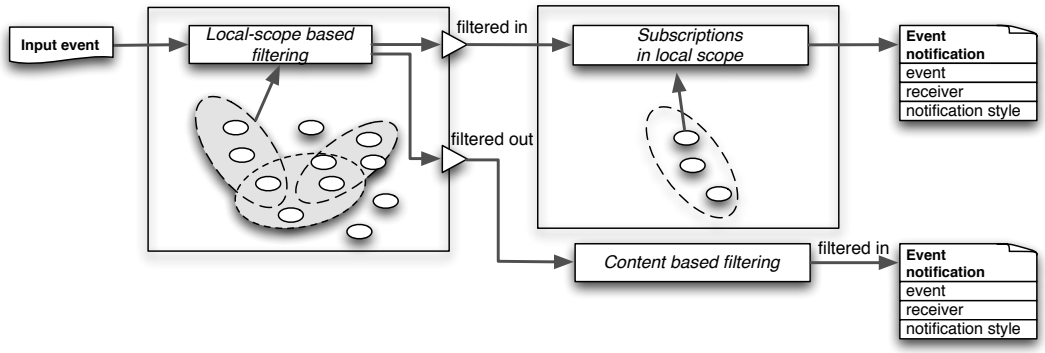


Figure 6.1. Local scope-based event filtering

The first step of filtering events based on an actor's local scope is very similar to the topic-based methods, as it introduces a programming abstraction to divide the event space into multiple sub-spaces. Then the filtering process is to determine whether the event falls into any of these sub-spaces. However, the major difference is that, while the concept of topic is usually static, the local scope is a dynamic concept. As we define the local scope as the actions in the field of work that an actor imposes certain level of intention or capability (Section 5.1.2), it is likely that it will be modified as the actor works on different tasks or the environment changes. The knowledge updating process as we describe in Section 5.4 allows the system to ensure that its representation of actors' local scopes is always updated to current state. As a result, the events that pass the first step are always relevant to the actor's current working context.

The second step in our method allows the users to further specify how they want to be notified for these events within their local scopes. Because the events that are passed into this step are limited to those within each actor's corresponding

local scope, the total number of events that each actor needs to manage is greatly reduced.

Considering the knowledge updating process mentioned in Section 5.4, not all the events can be successfully associated with entities in human actors' local scopes by the computer system. The system may not be able to recognize the relevance of some external events during the association and assessment steps at first, but rather depends on the human actors to interpret them and generate internal events that can be recognized later. For these events, we should still allow the users to subscribe to them using the content-based mechanisms. Because the events that need to be managed using the content-based mechanism are only a small subset that cannot be associated with local scopes, the overhead on the users to manage them are minimized.

In the following of this section, we focus on the local scope-based filtering and subscription mechanism. We first describe the algorithm to perform the local scope-based filtering in the first step, and then discuss the management of subscription within the local scopes. The content-based notification mechanisms have been well discussed in the literature on event-based computing [37], and many existing event infrastructures provide support to it, so we do not provide detail here. Readers can refer to [67] for more discussion.

6.1.1 Filtering events by local scopes

To filter events based on local scopes, we need to make use of the system generated knowledge attached to each event during the knowledge updating process, i.e. before an event is sent to the notification component, it should have been processed to update the system knowledge following the steps described in Section 5.4. As a result, every input event in the filtering operation is already transformed into an *event chain* $EC = (e_0, e_1, e_2, ...)$ that includes at least one original event e_0, and possibly additional derived and anticipatory events. If any event in the event chain has references to entities in the PlanGraph model, such references should have been established during the knowledge updating process.

Therefore, filtering the event by local scopes is the operation that takes an event chain EC and an actor ar, and then finds whether any event instance in the

event chain is relevant to the user based on ar's local scope. The operation can be performed in the following steps:

1. The local scopes of the actor ar is constructed from the current PlanGraph PG, following the algorithm described in Section 5.2.3.

2. For each event instance in the event chain EC, we first iterate through its attributes to check whether it has references to entities in the PlanGraph model. If none is found, the event instance is skipped.

3. If the event instance is found to have references to entities in the PlanGraph model, we check whether any entity is inside the local scope of ar. If so, we can claim that the event instance passes the filter.

The process can be described as a procedure $FILTER\text{-}LS$ as follow.

```
1: procedure FILTER-LS(EC, ar)
2:     LS = BUILD − LS(ar, PG)                          ▷ construct the local scope
3:     for all e in EC do
4:         for all attr in e.payload do
5:             if typeOf(attr) == EntityReference then
6:                 for all entity in LS do      ▷ each entity is a tuple of {ar, act, int}
7:                     if entity[1] == attr then                    ▷ A match is found
8:                         return {e, ar}
9:                     end if
10:                end for
11:            end if
12:        end for
13:    end for
14:    return null
15: end procedure
```

6.1.2 Managing subscriptions in local scopes

Once an event passes the local scope-based filtering, the corresponding actor's subscriptions within the local scope are applied to decide on whether and how the event should be notified.

Each subscription in the local scopes is defined as the pair of an event pattern and a notification style: $sub = (pat, ns)$, where the event pattern pat is specified by means of one or more user-defined filter expressions. Each filter expression takes the form of a predicate that is evaluated against an event. The event passes the filter if the predicate evaluates to $TRUE$, and fails the filter if the predicate evaluates to $FALSE$. Within an actor's local scope, three kinds of filter expression can be attached, and it is possible to mix these different kinds of expressions in one single event pattern:

1. An *event type* filter expression lists one or more event types. The expression evaluates to TRUE if the incoming event is an instance of any of these types.

2. An *event content* filter expression is evaluated from the values of payload attributes of the event instance. For example, $(attrA == valueX)$ and $(attrB > valueC)$.

3. A *local scope* filter expression is evaluated based on the different intention and capability levels of entities associated with the event. For example, the user may specify an event pattern to match all the events attached to actions that the user is *intended to* perform and are *workable*.

While the event type and content filters can use the same syntax as existing content-based filter expression languages, but the local scope filter expression is unique in our approach and provides an additional level of expressiveness than merely content-based subscriptions. It allows the human actors to describe event patterns based on the relations between the actor and the entities in the local scope, instead of the attributes of an event directly. It is quite common that the actor wants to treat events related to all the actions he/she has certain level of intention in the same way, but cannot specify what exactly these actions are as their local scopes are changing.

Along with each event pattern, the user needs to select a notification style for it. It has to be considered that there is a multitude of different notification styles with different trade-off between its potential to attract attention and its obtrusiveness [81]. Generally, we defined five notification styles.

1. The first style is actually not to present the notification at all. This provides another level of control for the users to filter out some events even when they fall into their local scopes.

2. The object-coupled notification presents some of the event attributes (e.g., time stamp or type) as graphical attributes of the icon representing the entity which is affected by the event. This provides less obtrusiveness to the user but is not perceivable unless the entity is within the user's viewport.

3. The standalone event window presents a dedicated window to display all the event notifications. It allows the user to keep track of different incoming events at the same place, but loses the visual connections between the events and their references.

4. Non-modal global notification presents each incoming event at a dedicated corner for a period of time and then disappears. It is more perceivable to the users, but requires the user to respond in a limited time.

5. Modal global notification uses a modal dialogue box to present a detailed description of an event, and the user has to take an active action to confirm the reception of the event (e.g. click the 'close' button) before going back to previous view.

To support the management of subscriptions within the local scope, the computer system can provide an interactive subscription tool as presented in Figure (??). As the system maintains the knowledge representation of the field of work and each user's local scope, it is easy to generate a diagrammatic representation of the field of work with the user's local scope of work highlighted. Such a visual interface allows the user to perform several tasks to manage the subscriptions:

1. The user can easily recognize what actions the system believes are part of his/her local scope. In case the user believes the system's representation is incorrect or wants to make changes, he/she can directly manipulate the visual representation to add/remove entities from the local scope.

2. The user can click on each of the nodes in the visualization to review active filters on each node, or create new filters.

add the figure of the subscription interface. The discussion is incomplete as well.

6.2 Supporting Event Interpretation

In our approach, we use the term 'event interpretation' to include both the process of *comprehension*, i.e. understanding the meaning of perceived event within the context of a user's current goals and activities, and the process of *projection*, i.e. predicting the future states based on the comprehension. Although these two awareness processes have different characteristics at the conceptual level, they are usually inseparable from the user's perspective. The user can switch back and forth between comprehension and projection upon perceiving an event without realizing the difference between them. Hence, we design for supporting event interpretation as an integrated cognitive process that covers both comprehension and projection.

Generally, we consider the event interpretation process as an analytical reasoning process triggered by perceiving an event notification. It includes three interleaving tasks:

1. Understanding the meaning of an event within the context of its origin of occurrence. This includes identifying the objects or relations that are mentioned in the event, and understanding what aspects of the objects or relations are changed.

2. Assessing the impact of the event on the user's current work context. This includes identifying which part of the actor's work is impacted by the event and what the consequences are.

3. Predicting the future states on other part of the actor's work or other actors' work because of the dependencies among activities.

In our approach, this analytical reasoning process can be supported from two aspects:

One one hand, the system can perform the reasoning tasks for the user. Actually, as we can see in Section 5.4, some of these reasoning tasks have already been performed by the system during the knowledge updating process. During the

association step, the system attempts to identify the objects or relations that are mentioned in the event. In the *assessment* step, the system evaluates the consequences of the event on all the entities in the field of work. In the *propagation* step, the system also predicts the future states on other activities. Hence, the system can support the user by simply present its reasoning results. In this way, some of the high level cognitive tasks are transformed into low level perceptual tasks for the user.

On the other hand, the system can provide external representations of the contextual information to help the users perform these reasoning tasks. As these reasoning tasks are always performed within certain contexts, such as the context of an event's origin, the user's current work context, or the dependencies among activities, the visualization of the contextual information serves as an important cognitive resource for the users.

From the above discussion, we present an information visualization framework for supporting event interpretation with three different types of visual representations (Figure 6.2):

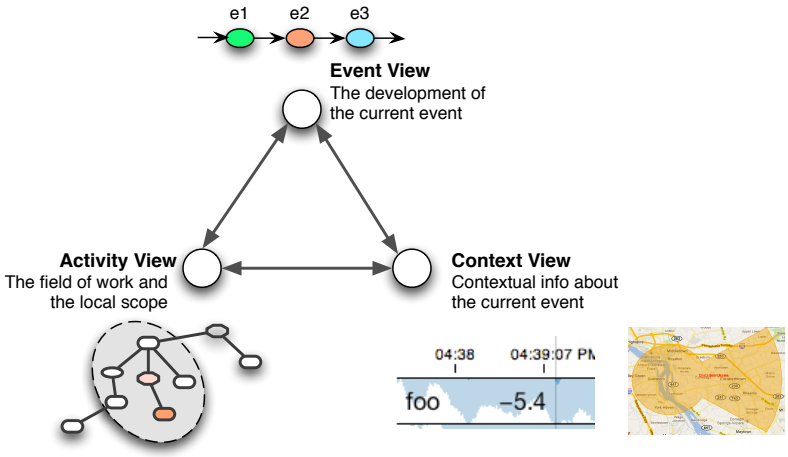


Figure 6.2. The visualization framework for event interpretation

1. Event view: visual representation of the event that needs to be interpreted.
2. Activity view: visual representation of the field of work and the actor's local scope.

3. Context view: visual representations of the contextual information that is necessary to interpret the event.

6.2.1 Event view

The event view is used to present the information about the event that needs to be interpreted. This includes not only the event that is notified to user, but also the whole event chain that the current event is situated in. As we defined in Section 5.4, an event chain records the development of an event in the knowledge updating process. In the assessment step, the system may generate derived events indicating the system's belief on how the field of work has been changed due to the original event. In the propagation step, the system predicates the future state changes, and add them as anticipatory events into the chain. In this way, the event chain actually represents the results of the system's reasoning on it.

Therefore, the event view is a diagrammatic representation including an ordered list of nodes (Figure 6.3). Each node represents an event in the corresponding event chain. The nodes are arranged in the development order of the events they represent. In a horizontal layout, the node at the most left represents the original event, and then is connected to the set of nodes representing the derived events, followed by the nodes for anticipatory events. The design detail about the event view is summarized as follow:

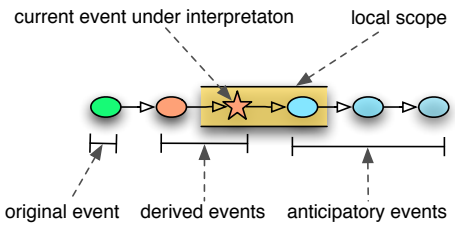


Figure 6.3. Visualizing the event chain in the event view

1. The node representing the current event that is notified to the user for interpretation is marked with a star, so that the user can know where the current event under interpretation is situated in the event chain.

2. The nodes representing events inside the user's local scope are shadowed so that the user can clearly see when the development of the event enters its work context.

3. The different types of events, i.e. original event, derived events, and anticipatory events, are distinguished by different colors.

4. We use different opacity levels to indicate the system's confidence level on the anticipatory events.

5. Clicking on the node will show the detail information about the represented event. If the event is associated with the objects represented in the context view or the entities represented in the activity view, they will be highlighted.

The event view plays two important functions during the event interpretation process. First, the event chain provides an overview of the system's reasoning process of an event, which allows the users to understand where an event comes from, and the possible consequences it leads to based on the system's reasoning. Second, it serves as a navigation interface that allows the user to click on nodes to check details on each step of the reasoning. In next section, we will introduce a revised version of the event view that can play the third function in event propagation to present the development of an event across multiple actors.

6.2.2 Activity view

The activity view provides an overview of the field of work and the user's local scope. Such a representation provides the information about the goals, actions, resources, and actors that are participated in the collaborative activity and the relations between these different entities. As argued by Carroll et al. [20], such an external representation of the overall situation is very important for awareness development when group work is distributed and many dependencies exist in the collaborative activity. Within the context of event interpretation, we believe the activity view can provide several kinds of support to the user:

1. It provides the activity-related frame of reference to understand an event and the system's reasoning on it. The activity view is linked to the nodes in

the event view. Whenever the user looks at an event node that is associated with an entity in the field of work, the corresponding object in the activity view is highlighted. Thus, the user can perceive which part of the activity is impacted by the event.

2. It provides the contextual information from the perspective of the collaborative activity that is necessary for the user to perform reasoning. The activity view allows the user to navigate through the different entities in the field of work, check the detail on each entity, and recognize the different relations between them, which is very important for the user to evaluate and predict the impacts of an event.

3. It shows the boundary between the entities inside and outside the user's local scope, so that the user always know what part of the field of work he/she needs to focus on.

The activity view consists of a diagrammatic representation of the entities and relations in the field of work, and an interaction interface to navigate through the view. The generation of the activity view is straightforward in our approach. As we maintain the knowledge representation of the whole field of work in the PlanGraph model, it can be easily externalized into a graph visualization with the entities as nodes, and the relations as links. The interaction interface allows the user to perform actions such as zoom in/out, move the viewport, check details, and dynamic queries.

A key issue in designing the activity view is the problem of *discernibility* in graph visualization when the number of elements is large [53]. It is well known that comprehension and detailed analysis of data in graph structures is easier when the size of the displayed graph is small. Displaying an entire large graph may give an indication of the overall structure, but makes it difficult to comprehend the details.

To address this problem, we treat the entities inside and outside the local scope differently when visualizing them in the activity view (Figure 6.4):

1. The entities within the local scope are visualized as larger nodes that allows to encode more detailed information, such as the name, type, and execution

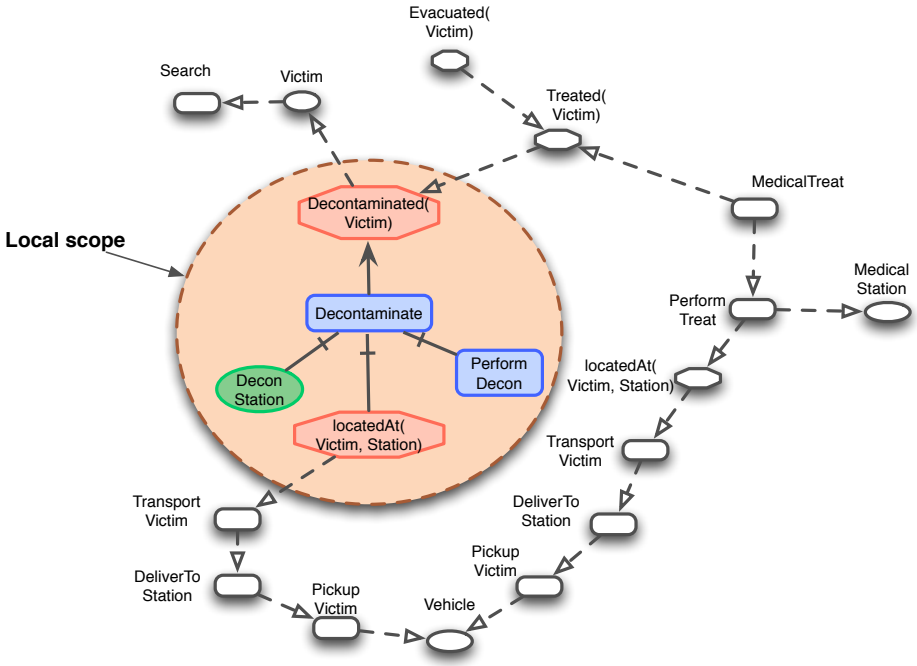


Figure 6.4. Visualizing the field of work in the activity view

state. But the entities outside the local scope are visualized as small icons that have less information encoded.

2. The different relations between entities with in the local scope are considered and visualized using different types of line styles, but the entities outside the local scope are connected merely based on the dependencies.

3. The entities in the local scope are visualized using the hierarchical layout that emphasizes the decomposition relations among actions. The entities outside adopt a more space-efficient graph layout (e.g. force-directed layout [41]) to show a maximum number of nodes within the limited view port.

The distinction between the entities inside and outside the local scope in the activity view complies with the distributed nature of the field of work. As for each actor, the local scope is where he/she possesses more control and also is the most likely place where the event interpretation will occur. As a result, the user would like to see more details on the entities within the local scope. Meanwhile, the entities outside the local scope are distant from his/her work focus, and all

he/she needs to know is how these entities are related to the local scope, with less detailed information about them.

6.2.3 Context view

The context view is a container for interactive information visualization tools to understand the different contexts of the event, including the the context of an event's origin, the user's current work context, or the dependencies among activities. The contexts emerge from a complex mix of entities in the field of work, including the objects that the user is working on, the collaborating actors, resources, and informational objects that are necessary to understand the situation. In relative simple collaborative environments, such as the collaborative editors, the context can be represented as one single shared workspace. But in complex, real-world collaboration, the contextual information is usually in diverse forms and should be understood from different perspectives. As a result, the context view is usually a set of coordinated and multiple views to represent different perspectives of the context. For example, it may include a cartographic map to display the distribution of actors, a data table to list all the available resource, a time line to show the historical usage of a resource, as well as other diverse visual displays including scatter plots, parallel coordinate plots, histograms etc. The set of views and interactive tools that are needed usually depends on the application domain and each actor's task in the domain.

6.3 Mediating Event Propagation

Event propagation refers to the social process to establish compatible awareness across multiple actors. As we argue in Section 2.4, an essential aspect of the awareness phenomena in distributed, complex collaboration is that the development of awareness should be considered as a social process that can involve a series of interactions among multiple actors. With aspect to the event-driven awareness process, that is to say the awareness process triggered by one event can be developed by multiple actors. The actor who receives the initial event generates his/her own interpretation of the event, and then externalize the interpretation as a new event,

which is then received by the second actor. The second actor's interpretation is built on top of the first actor's, and may generate new events that are received by other actors. In this way, as the initial event is propagated to multiple actors, the team awareness is developed.

Unlike the event notification and event interpretation that are operated on each single event, each event propagation process can be considered as a meta-process that consists of multiple processes on multiple events. Figure 6.5 shows an example of the event propagation process that involves the computer system and three human actors. In the beginning, an input event e_0 is received by the system and the system derives a new event e_1 during its knowledge updating process. This derived event e_1 indicates a state change of an action that falls into $actor1$'s local scope, hence it will be notified to $actor1$ during the notification process. Upon receiving the notification, $actor1$ performs his/her own interpretation process, and externalize the result of the interpretation as a new event e_2. e_2 is sent back to the system during the externalization process. After receiving e_2, the system starts another round of knowledge updating and notification, and finds that e_2 falls into $actor2$'s local scope. As a result, the system notifies $actor2$ about e_2. The process continues as $actor2$ further interprets the event and generates new event, which is later notified to $actor3$.

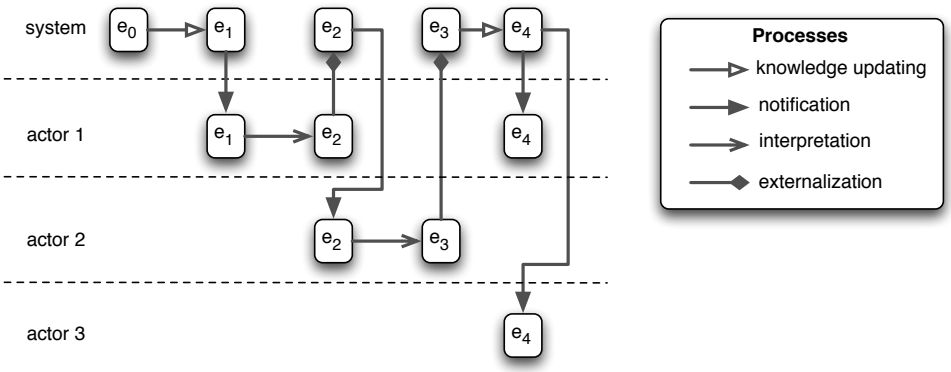


Figure 6.5. An example of the event propagation process

From the example, we can clearly see that the event propagation is achieved by the joint effort of the human actors and the computer system. One one hand, it relies on the human actors to interpret the events built on top of other's inter-

pretation, and externalize their own interpretations as new events. On the other hand, the system should be able to disseminate these new events to the actors who are interested in further developing them.

The capability of system to disseminate these derived events to relevant actors can be achieved through the local scope-based event notification mechanism as we described in Section 6.1. Because the relevance of events to the actors are judged by whether they are associated with the entities in the actors' local scopes, the actors do not need to subscribe to them in advance. As long as the new events generated by other actors fall into my local scope, I will get notified.

more discussion on how the propagation works: the basic assumption is the connectivity of local scopes. through shared entity and dependencies.

Beyond that, the system can mediate the event propagation from the human actor's perspective, i.e. to help the human actors do their jobs in propagating events. The human actors have to be enabled to:

1. keep track of the historical development of the event so that they know where an event comes from, how it has been developed, and the actors who have contributed to its development.

2. easily externalize their interpretation of the event, i.e. how their intentions and beliefs are changed because of the event.

3. control the visibility of their newly generated events, such as who should be able to receive their interpretation, and to what level of detail.

The following of this section focuses on how these tasks can be achieved by human actors with the computer system's assistance. It requires several enhancements in the knowledge updating process, the event notification mechanism, and the visualization framework for event interpretation, so that the social aspect of the event propagation is explicitly represented and supported.

6.3.1 Tracking event propagation

In order to keep track of the event propagation, we extend the concept of *event chain* to the concept of *event propagation tree*, and thereafter revise the event view in the visualization framework to represent the event propagation tree.

In Section 5.4, we introduce the concept of *event chain* to record the system's development of an event in the knowledge updating process. Now, we extend it to the concept of *event propagation tree* to keep track of the event propagation process from the following two aspects:

1. As the event propagation is a meta-process that can involve both the computer system's knowledge updating process and human actors' awareness processes, the events in an *event propagation tree* include not only the events generated in the system's reasoning process, but also those generated by the human actors in the externalization process.

2. In the event propagation process, it is possible that one event is interpreted by multiple human actors, along with the system's reasoning on it. One event can then be derived into multiple branches, each of which can be further developed into multiple subsidiary branches. As a result, an *event propagation tree* EPT can no longer be represented as a linear sequence of events, but rather a tree structure (Figure 6.6). The root node in the tree structure e_0 represents the original event that triggers the propagation, and each node can have multiple branches indicating how the event represented by the node is further developed.

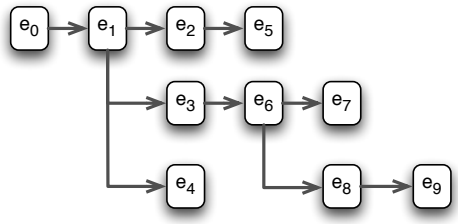


Figure 6.6. The event propagation tree

Similar to the *event chain*, the structure of an *event propagation tree* can be represented in our event representation by adding attributes in the *open content* section of each event (Figure 6.7):

1. A text string (*label*) indicates the functional role of the event in the event chain, i.e. whether the event is an *original* event, a *derived* event, or an *anticipatory* event.

2. An event reference (*parentEvent*) points to the parent event where the current event is derived from.

3. A list of event references (*childrenEvents*) points to all the event that are derived from the current event.

In addition, the *event source* attribute in the *header* section indicates the producer of each event, which can be the computer system or one of the human actors.

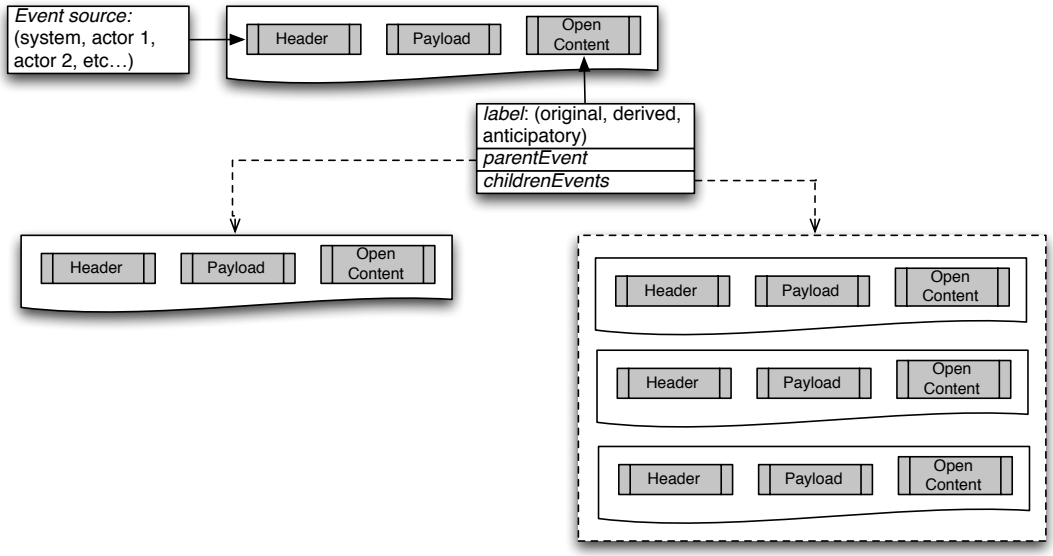


Figure 6.7. Event structure in an event propagation tree

By recoding the event propagation tree in the system's knowledge representation, the system can help the human actors to keep track of the event propagation by providing an extended version of the event view in the visualization framework to represent the event propagation tree. Comparing with the original event view, this extended event view provides an overview of the whole event propagation process, not only the results of the system's reasoning, but also how the other actors have contributed to it.

Initially, the extended version of the event view uses a history tree representation [96] to visualize the whole structure of an event propagation tree (Figure 6.8(a)). The event propagation tree is drawn using a right heavy horizontal-vertical

tree layout. A horizontal link indicates that the event on the right is derived from the left one. A new branch is created below existing ones in the vertical direction.

To understand the social context of the event propagation, it is important to see the actors who are participated in the development process and distinguish the events generated by different actors. In order to achieve this, we adopt a complementary layout that divides the visualization space into multiple horizontal rows, and each row only contains the events generated by a particular actor. However, presenting the whole event propagation tree in this layout can cause crossing edges that undermine the discernibility. Hence, we only use this layout to visualize the *active event propagation chain*. We define the *active event propagation chain* as the path from the root node of the event propagation chain to the node that represents the current event under interpretation. Figure 6.8(b) shows an example of visualizing the active event propagation chain in this layout. The user can toggle between these two representations via the interface.

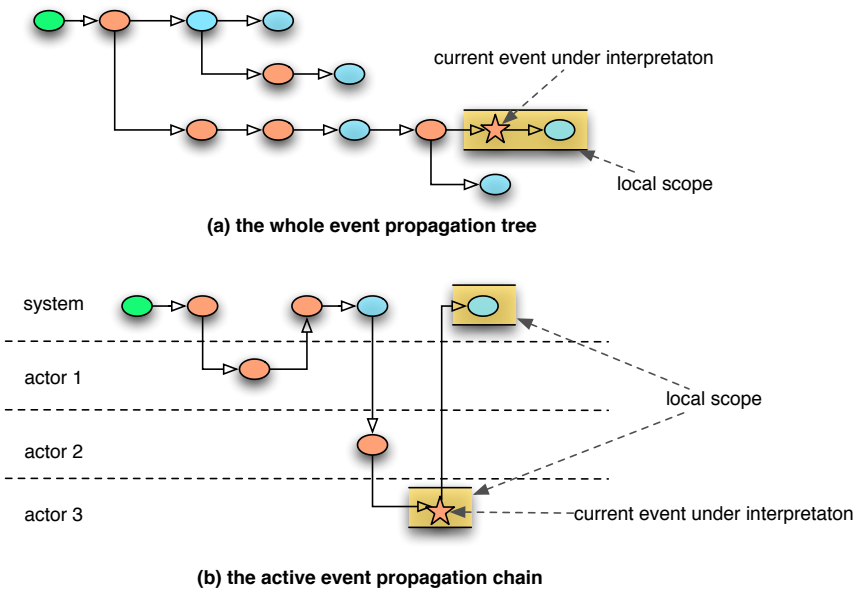


Figure 6.8. Visualizing the event propagation tree in the event view

6.3.2 Supporting externalization

Externalization is the process that the human actors express the results of their interpretation of an event as internal events and send them to the computer system

for further processing. An important characteristics of these internal events is that they are always associated with certain entities or relations in the PlanGraph model, as they reflect the changes of human actors' internal mental states in the field of work. As a result, we support the externalization by allowing the human actors to directly manipulate the nodes in the activity view where they can create internal events expressing their intentions or beliefs on corresponding entities in the field of work.

Whenever the human actor selects a node in the activity view, the option to create new events will become available in the interface (e.g. in the context menu or toolbar). The types of events that a human actor can generate on each node depends on the type of the corresponding entity in the PlanGraph, i.e. whether it is an action, a parameter, or a condition, and whether it is inside the human actor's local scope or not.

For an action node inside the local scope, the human actor can generate the following types of events:

1. *Intention events.* The human actor can create intention events to indicate the change of intention type (i.e. $Pot.Int$, $Int.Th$, or $Int.To$) toward the selected action.

2. *Belief events about the capability.* The human actor can create belief events to indicate the change of his/her capability (i.e. $Knows$, $Able$, or $Workable$) of performing the selected action.

3. *Belief events about the execution states.* The human actor can also create belief events to indicate his/her belief in the execution state change of the selected action.

4. *Belief events about the plan.* The human actor can also create belief events to modify the current plan of an action. On one hand, the human actor can indicate that he/she believes that the selected action is not part of the plan to performing the parent action, i.e. the current node should be deleted from the PlanGraph. On the other hand, the human actor can elaborate the selected action by adding a new parameter, condition, or subsidiary action to it.

For the parameter nodes inside the local scope, the human actor can also generate belief events about the execution states or contribute to the plan in the similar way as action nodes. However, an additional type of event is available for the parameter nodes, that is the human actor can express the assignment of a particular resource to the selected parameter as a belief event. To allow the human actor to externalize such type of events, a data table containing the list of available resources for the parameter will be displayed so that the human actor can select from the list.

For the condition nodes inside the local scope, belief events about the execution states or plan are also available. Besides, the human actor can create belief events to indicate the modification of the variables in a condition expression. For example, for a condition node representing the temporal constraint on a delivery action, i.e. the delivery must be completed before a given time, the user can select the node and create new event to modify the delivery time.

For the action nodes outside the local scope, only the intention events and belief events about the capability are available, as the human actor does not have control on the actions that are out of his/her local scope. However, the intention events and belief events about the capability provide a way for the human actor to expand his/her local scope to the outside nodes.

Table 6.1 shows a summary of the different types of events that can be externalized on each type of node in the activity view.

When the human actor generates an event, he/she is asked to provide a free-text explanation of this event, and to indicate whether the event is about the current state of the field of work (i.e. a derived event), or an anticipatory event about future states. If the event is a belief event, he/she also needs to give a confidence level to describe how confident he/she is about this event. When the new event is submitted to the system, a reference to the current event under interpretation will automatically attached to this new event so that the system can insert it to the event propagation tree.

Node type	Available event types for externalization
Action node inside the local scope	1. Intention events 2. Belief events about the capability 3. Belief events about the execution states 4. Belief events about the plan
Prameter node inside the local scope	1. Belief events about the execution states 2. Belief events about the plan 3. Belief events about value assignment
Condition node inside the local scope	1. Belief events about the execution states 2. Belief events about the plan 3. Belief events about variable modification
Action node outside the local scope	1. Intention events 2. Belief events about the capability

Table 6.1. Different event types for externalization

6.3.3 Controlling visibility

As we allow the human actors to externalize their interpretation as new events, we need to provide the interface to allow them to control the visibility of their events. It is very common that the human actors only want to share certain events with a subset of collaborators, or only showing a portion of the attributes available in the event structure. However, specifying the visibility on each individual event whenever it is externalized can be time consuming and therefore discourage the human actors to contribute. To address this problem, we utilize the local scopes to allow the human actors to specify the policies for controlling visibility in an abstract level, similarly as they manage subscriptions for the event notification.

In general, we define each visibility policy as a tuple of an event pattern, a receiver pattern, and a list of visible attributes: $vis = (ep, rp, attrs)$. The event pattern ep is similar to the same concept in an event subscription (Section 6.1.2), and is used to select the subset of events whose visibility will be regulated by this policy. The receiver pattern rp is used to select the subset of human actors to whom the selected events are visible. The attribute list $attr$ is a subset of the attributes of the selected events that are visible.

The event patterns are specified in the similar way as in an event subscription, by means of one or more event filter expressions that are evaluated against an event.

The receiver pattern is specified by means of one or more filter expressions that are evaluated against an actor. In general, we define two types of filter expressions that can be used to define a receiver pattern.

1. An *attribute* filter expression is evaluated from the attribute values of an actor. For example, it can filter the actors by their names, roles, or current locations.

2. A *local scope* filter expression is evaluated based on the different intention and capability levels an actor has on the same entity associated with the event. For example, the user may specify an event associated with an action is only visible to the actors who intend to perform the action, or the actors who are capable of performing the action.

To control the visibility as defined in the visibility policies, the event notification mechanism needs to be extended so that the visibility of an event is taken into consideration before it is notified to an actor. This is achieved by adding an extra step $APPLY\text{-}VIS$ after the event filtering to apply the set of visibility policies associated with actor who generated this event. The $APPLY\text{-}VIS$ operation takes an event notification en with the information about the corresponding event e and the receiver rec as the parameter, and decide on whether the notification should be delivered and whether the attributes need to be filtered out based on the visibility policies. In general, it can be performed in the following steps:

1. The actor who generated the event e is identified based on the *event source* attribute, and the set of visibility policies V are loaded.

2. For each visibility policy vis in V, we first check whether it satisfies the event pattern ep described in vis. If it does not, we repeat this step on the next policy in V.

3. If the event e satisfies the event pattern in vis, we check whether the receiver rec satisfies the receiver pattern rp described in vis. If it does not, it means rec should not be able to receive this notification, so we end the process.

4. If the receiver rec does satisfy the receiver pattern rp described in vis, we modify the event notification en to only include the attributes listed in the

attribute list *attr* described in *vis*. Then we repeat the second step on the next policy in V.

The *APPLY-VIS* process can be described as follow:

```
1: procedure APPLY-VIS(en)
2:     e = en.event
3:     rec = en.receiver
4:     V = LOAD-VIS(e.header.EventSource)      ▷ load the set of visibility policies
5:     for all vis in V do
6:         if SATISFY-E-PATTERN(e, vis.ep) then
7:             if SATISFY-R-PATTERN(rec, vis.rp) then
8:                 if en.attrs.ISEMPTY then
9:                     en.attrs = vis.attrs
10:                else
11:                    for all attr in en.attrs do
12:                        if attr not in vis.attrs then
13:                            en.attrs.REMOVE(attr)
14:                        end if
15:                    end for
16:                end if
17:            else
18:                return FALSE
19:            end if
20:        end if
21:    end for
22:    return TRUE
23: end procedure
```

6.4 Discussion

In this chapter, we describe our approach to promoting the event-driven awareness in the whole awareness development cycle:

1. The local space-based event notification mechanism is used to support the event perception by filtering out the irrelevant events and deciding on the

notification styles based on the knowledge about the local scope and event subscriptions.

2. The event comprehension and projection are supported by the visualization framework for supporting event interpretation that we define as an integrated, analytical reasoning process that include both comprehension and projection.

3. The social process to establish compatible awareness across multiple actors is mediated by a set of computational and interactive tools to support event propagation.

The design of these components follows the two design principles of awareness promotion (Chapter 4), i.e. the utility of the computer's knowledge representation and the divisions of responsibility between the computer and human actors, to address the challenge when both the complexity and dynamics scale up in collaborative activities. As a summary, Table 6.2 shows how these awareness promotion components in our approach follow the two design principles, and how the problem of scaling up is addressed.

Aspects	**Event notification**	**Event interpretation**	**Event propagation**
Use of knowledge representation	The local scopes are used to filter events	1. The event chain is used to visualize the event view 2. The PlanGraph, local scopes, and dependencies are used to visualize the activity view	1. The event propagation tree is used to visualize the event view 2. The PlanGraph is used to support externalization 3. The local scope is used to manage visibility
Computer's responsibility	1. Perform the filtering based on local scopes and subscriptions 2. Visualize the local scope for the users to manage the subscriptions	1. Perform reasoning on the event and present the results as the event chain 2. Visualize the field of work and the local scope 3. Provide external representations of the contexual information	1. Keep track of the event propagation tree, and present it to the users 2. Provide the interface for the user to externalize the results of interpretation. 3. Apply the visibility policies for notification

Human actor's responsibility	1. Specify the event interests as subscriptions 2. Modify the local scope	1. Navigate through the event view to review the system's reasoning 2. Interpret the event within the activity view and context view	1. Contribut to the event propagation by externalization 2. Control the visibility by specifying visibility policies
Reduce complexity	The number of subscriptions that the human actor needs to managed is limited in the local scope	1. The system performs the reasoning tasks so that some of the high level cognitive tasks are transformed into low level perceptual tasks for the user. 2. The activity view shows more details for the entities in the local scope, but reduce details for outsider.	1. The event propagation tree is automatically recorded by the system, so the human actor does not need to maintain it in the mind. 2. The externalization can be performed directly on the node 3. The visibility control is specified in relation to local scopes, instead of on each event.
Handle dynamics	The filtering is built on top of the local scope that is dynamically constructed	1. Each event is consumed by the system to update the PlanGraph model 2. The activity view reflects the current state of the field of work	1. The event propagation chain records the historical development of an event. 2. The visibility is built on top of the local scope that is dynamically constructed

Table 6.2: Summary of the event-driven awareness promotion

Chapter 7

Architecture and Implementation

To validate our computational approach, a prototype system EDAP (Event-driven awareness promotion) is implemented in this study. EDAP server provides a generic event-driven infrastructure for awareness promotion that can be applied in different application domain. EDAP client is a standalone Web-based interactive awareness system that is used to demonstrate the functionality of our approach. In this chapter, we first describe the architecture of the system. Then we describe the implementation details of the server and client.

7.1 Architecture of EDAP

EDAP follows a typical client/server architecture model (Figure 7.1). The EDAP server provides the major functions of the event-driven awareness promotion, i.e. it maintains the knowledge representation, performs the knowledge updating and event processing. The EDAP client provides the visual and interactive environment that allows the user to perceive and interpret events, generate new events, and manage subscriptions and visibility policies.

The interaction between the client and server follows the service-oriented network protocols. The REST protocol is used to implement the request/response communication style, i.e. clients initiate requests to servers and servers process requests and return appropriate responses. The COMET protocol is used to offer the push notification from the server to the client by maintaining a long-held HTTP request between them.

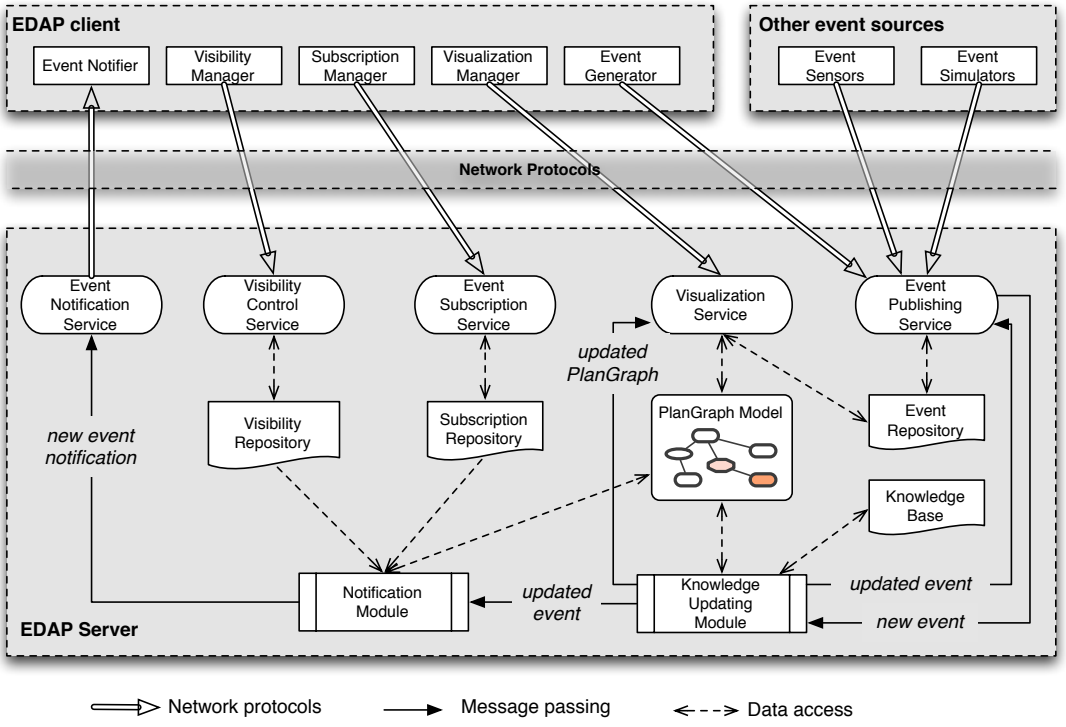


Figure 7.1. Architecture of EDAP

The EDAP client provides several components that can interact with the server. The *Event Generator* is used to create new events in the externalization process, and post the new events to the server for processing. The *Visualization Manager* requests visualization specifications about the event view, activity view, and context view from the server, and render them in the client's interface. The *Subscription Manager* requests the existing subscriptions from the server, provides the interface for the user to manage them or create new ones, and then send the modifications back to the server. The *Visibility Manager* provides the similar function as the *Subscription Manager* to manage visibility policies. The *Event Notifier* establishes a long-polling connection with the server. Whenever a new event notification is sent from the server, it applies the corresponding notification style to notify the user.

To respond to client requests, the EDAP server provides several components in the form of web services to handle the different types of requests. The *Event Publishing Service* responds to the new events posted from the client or other event

sources, stores them in the event repository, and notifies the *Knowledge Updating Module* to process them. The *Visualization Service* generates the visualization specifications for the event view and activity view based on the knowledge from PlanGraph model and event repository. The *Event Subscription Service* responds to the *Subscription Manager* and updates the subscription repository. The *Visibility Control Service* responds to the *Visibility Manager* and updates the visibility policy repository. The *Event Notification Service* responds to the message passed from the *Notification Module* whenever a new event notification is generated, and push it to the corresponding clients. Besides these service-oriented modules, the server also includes two important processing modules: the *Knowledge Updating Module* perform the knowledge updating as described in Section 5.4 whenever new events are received; and the *Notification Module* provides the local scope-based event notification mechanism (Section 6.1) to distribute events.

The communication between modules in the EDAP server is achieved through a limited form of the publish/subscribe interaction. Each module can publish important messages during its performance, and the other modules who need to respond to these messages need to subscribe to them. In this way, the interaction between modules can be asynchronous so that they do not block each other while waiting for others to finish their task. In addition, it provides a distributed interaction style that allows the different modules to be deployed across multiple machines.

7.2 Implementation of EDAP Server

The functional modules EDAP server is implemented using the Python programing language, and the web services are developed in the Django framework. The three data repositories, i.e. event, subscription and visibility repositories, as well as the knowledge base, are implemented as PostgreSQl databases. In the following, we describe the implementation details on major modules and the repositories are discussed within the respective modules that can manipulate them.

7.2.1 Service-oriented modules

The EDAP server includes five service-oriented modules to handle the different types of client requests.

Event Publishing Service The *Event Publishing Service* is the first responder to the new events that are published to the system, and its major responsibility is to store the events in the event repository. There are two types of event sources where the *Event Publishing Service* can receive events:

1. Events can come externally from the different types of clients as HTTP requests, including the EDAP client or other event sensors or simulators. For these events, after storing them in the repository, the service also needs to send out a *'NewEvent'* message notifying the *Knowledge Updating Module* to perform knowledge updating on the new event.

2. They can also come from the *Knowledge Updating Module* as it generates new derived or anticipatory events. These events are published by the *Knowledge Updating Module* as *'UpdatedEvent'* messages, and the *Event Publishing Service* needs to subscribe to them and store them in the repository.

The events received by the *Event Publishing Service* are described as attribute-value tuples, and the service needs to store these events in the event repository. The event repository needs to be semi-structured as the different event types supported by the system can have different set of attributes. However, this cannot be directly achieved in a relational database like PostgreSQl. As a result, we only store the header of each event directly in the data tables, but add two additional long text field (*payload* and *opencontent*) for each record. These two fields store the XML formatted tuples corresponding to the attributes in the event's payload and open content. Besides the event table that stores all the events, the event repository also includes a relation table to record the event chains or event propagation trees. Whenever an event is added to the repository, a new record is added to the event development table as well with the id of parent event (null for original events) and the current event, along with the label indicating the type of the event. Figure 7.2 shows the data tables in the event repository.

Figure 7.2. Data tables in the event repository

Visualization Service The *Visualization Service* responds to the client's request for visualizing the event view or activity view. The service retrieves the necessary data from event repository or PlanGraph model, uses it to generate a visualization specification in the JSON format and sends it back to the client.

For the request to visualize the event view given the current event under interpretation, the service searches the event repository recursively to find all the events that are included in the event propagation tree, and generates a JSON object to represent the tree structure.

In order to generate the visualization specification for the activity view, the service maintains the most recent version of the local scopes and dependency network constructed from the PlanGraph model. The *Visualization Service* subscribes to the message *'updatedPlanGraph'* that is published by the *Knowledge Updating Module* each time the PlanGraph model is changed. Hence, whenever the *Visualization Service* receives such message, it uses the PlanGraph model to reconstruct the local scopes and dependency network so that it always keeps the newest version. Whenever the client requests an activity view given a specific user, the user's local scope is first retrieved, and the entities in the local scope are added to the visualization specification. Then the entities in the local scope are searched in the dependency network recursively to find all the other entities that have dependency relations with them, and these entities outside the local scope are added to visualization specification.

Event Subscription Service The *Event Subscription Service* responds to the client's request to list all the current subscriptions, add a new subscription, modify or delete an existing subscription. The existing subscription of each user is stored in the subscription repository. If the request is to add a new subscription, a new record will be added into the repository. If the request is to modify or delete an existing one, the subscription id will be used to identify the record in the repository and perform the update or delete operation.

Because the event pattern in each subscription is semi-structured with multiple filter expressions, we also use the XML formatted text field to store the filters for each subscription. Figure 7.3 shows the subscription table with an example of the XML formatted event pattern that include all the three types of filter expressions discussed in Section 6.1.2.

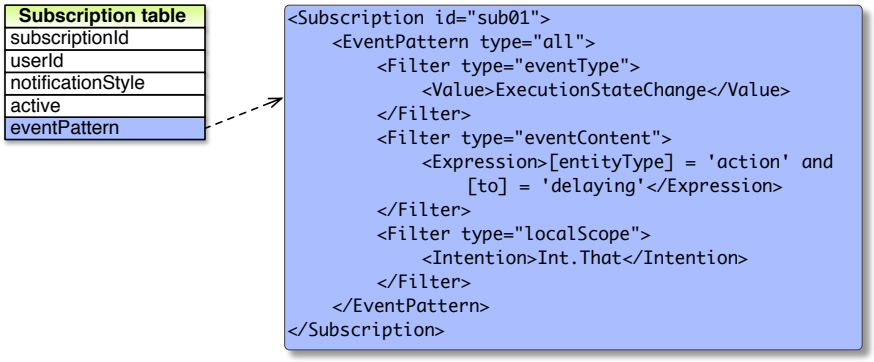


Figure 7.3. An example subscription record

Visibility Control Service The *Visibility Control Service* works in the similar way as the *Event Subscription Service*, to manipulate the visibility repository to retrieve, add, modify or delete visibility policies. Unlike the subscriptions, each visibility policy has three semi-structured fields: the *eventPattern* is similar to the event pattern in subscriptions, the *receiverPattern* includes the filter expressions that are used to define the receivers to whom the event is visible, and *attributeList* is the list of visible attributes. Figure 7.4 shows an example of a record in the visibility repository.

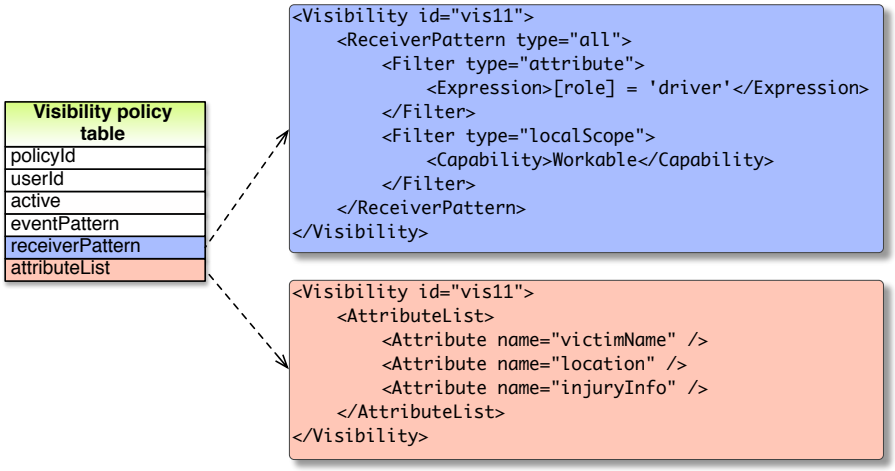


Figure 7.4. An example visibility policy record

Event Notification Service The *Event Notification Service* maintains a long polling connection with each client so that it can push new event notifications to the client. The service stores a routing table that maps each user id to the corresponding client connection. The service subscribes to the message *'newEventNotification'* that is published by the *Notification Module* each time a new event notification is generated. Whenever the *Event Notification Service* receives such a message, it searches the routing table to find a match between the receiver of the notification and the client connection. If a connection is found, the service pushes the event notification to the corresponding user via the client connection.

7.2.2 The Knowledge updating module

The *Knowledge Updating Module* performs the knowledge updating process as we described in Section 5.4. As the *Knowledge Updating Module* manipulates the PlanGraph model, we first describe how the PlanGraph model is implemented, then discuss the implementation of the knowledge updating process.

Implementation of the PlanGraph The PlanGraph is implemented in the object-oriented paradigm as a dynamic data structure by several data objects. The overall *PlanGraph* object points to the root plan node. Each *PlanNode* objects includes attributes recording the parent node and the subsidiary plan nodes, which

together allows the recursive traverse throughout the PlanGraph hierarchy.

The *PlanNode* has three types of subsidiary classes, corresponding to the three types of nodes in the PlanGraph model.

1. Each *ActionNode* defines the properties of an action, such as the type of the action, whether it is a basic or complex action, the execution state, the current recipe of the action. Each *ActionNode* also includes two slots to store the actors' intentions and capabilities towards the action. Besides, each *ActionNode* includes a list of parameters and the conditions.

2. Each *ParamNode* object represents a parameter that includes the type of the parameter, the execution state, the values, and the list of conditions about the parameter.

3. Each *CondNode* object represents a condition that includes the type of the parameter, the execution state, and an expression indicating the content of the condition.

Implementation of the knowledge updating process The *Knowledge Updating Module* includes four subsidiary modules, complying with the four steps of the knowledge updating process.

1. *Association* sub-module establishes the association between the input event and the current PlanGraph mode by searching all the nodes in the PlanGraph. It starts with the recognition of event type for each input event, as the different event types are treated differently. The association is implemented as a native function in Python that follows the search strategies we discussed in Section 5.4.1. Once an association is found, it updates the value of the corresponding node in the PlanGraph, and then add the pointer to the node into the event object.

2. *Assessment* sub-module evaluates each event check how it can lead to changes towards the action performance, such as the execution state change of an action, or a condition. This sub-module is achieved by defining a set of assessment rules and employing a knowledge engine Pyke to enable the forward

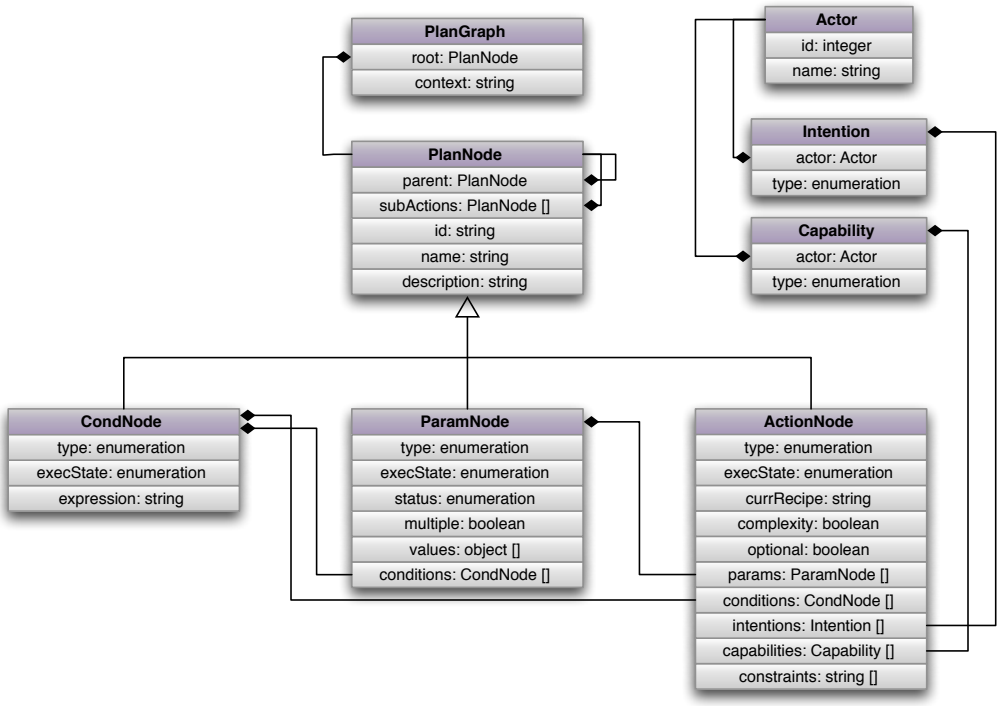


Figure 7.5. The class diagram of the PlanGraph

chaining inference. The knowledge engine uses the PlanGraph to assess all the possible changes due to the new information stored in the event, and then generate new events to represent these changes. Whenever a new event is generated in this step, the module publishes a new *'NewEvent'* message, so that the other modules who subscribe to it can get notified.

3. *Elaboration* sub-module advances the collaborative activity from the system side based on the change from the new event. The elaboration is achieved with the support of two types of knowledge stored in the knowledge base: (1) the recipe knowledge about how to derive an action into a sequence of parameters, pre-conditions, and subsidiary actions, how to identify a unbound parameter, or how to satisfy a condition, etc.; (2) the knowledge about actor roles and the potential intentions and capabilities towards actions. These two types of knowledge are stored in the *knowledge base* in the form of a PostgreSQl database. The content of each recipe is stored in an XML formatted text field along with other general information about the recipe. The

actor role specifications are stored in a data table mapping each role to the potential intended action type, or the actions that the actors in the role are capable of performing.

add an example recipe here.

4. *Propagation* sub-module is a standard Bayesian network-based reasoning process. This module first constructs the dependency network from the PlanGraph model following the algorithm described in Section 5.2.4. Then we employ the open source Python library, OpenBayes, to propagate the state change in the dependency network. Based on the results of the propagation, we generate new events to represent these changes, and new *'NewEvent'* messages will be published to notify other modules.

After the four steps have been performed, the knowledge updating module publishes two new messages *'UpdatedEvent'* and *'UpdatedPlanGraph'*. The former is used by the *Notification Module* to start generating event notifications, and the latter notifies the *Visualization Service* to reconstruct the local scopes and dependency network.

7.2.3 The Notification module

The *Notification Module* performs the event notification algorithms to generate event notifications after the knowledge updating process is done on each event, i.e. whenever it receives a new message *'UpdatedEvent'* from the *Knowledge Updating Module*. The *Notification Module* generates event notifications in three steps, and different type of knowledge is used in each step.

1. The PlanGraph is used in the first step to perform the local scope-based filtering as described in Section 6.1.1. The module implements the $FILTER\text{-}LS$ algorithm as a standard Python function and execute it on each user participated in the PlanGraph. If the function returns true, the second step is performed.

2. During the second step, the user's subscriptions are retrieved from the subscription repository, and the event is evaluated against each of the subscriptions. If the event matches the event pattern of a subscription, the third step is performed.

3. In this step, the user's visibility policies are retrieved from the visibility repository, and the $APPLY\text{-}VIS$ algorithm (Section 6.3.3) is implemented to apply the visibility policies to the event notification. If function returns true, i.e. the event is visible to the actor, then the corresponding event notification is generated.

Once an event notification is generated, the *Notification Module* publishes a new messages *'NewEventNotification'*, which is subscribed by the *Event Notification Service* who will push the event notification to the user's client.

7.3 Implementation of EDAP Client

The EDAP client is a Web-based client provides the user with the lightweight HTML + JavaScript user interface to the awareness system. The EDAP client is implemented by using several Open Source JavaScript libraries: JQuery framework for the interface layout, the OpenLayers mapping library to provide the geographic context view, and the d3 library to generate the event view and activity view. The thin client design allows the user to easily access the system without installing any software or plug-ins.

The EDAP client is developed for supporting awareness in the motivating scenario of emergency response described in Section 1.1.1. Within this scenario, the awareness process can involve multiple actors. Imagine the occurrence of an unexpected event indicating a traffic jam that blocks the route for delivering a victim to a decontamination station. This event is first perceived by the transportation manager, who interprets the event as the cause of a delay for a vehicle that is used to deliver a victim to the decontamination station as scheduled. As a result, the decontamination managers intention to have the victim delivered at the decontamination station on time becomes problematic, which is further interpreted as a cause for delay on the decontamination operation. Furthermore, the delay on decontamination could impact the victim manager due to the temporal dependency between decontamination and medical treatment. The victim manager may have to re-schedule the victim so that the medical resource could be assigned to another victim.

Figure 7.6 shows the main interface of the EDAP client from the victim managers perspective. We provide three inter-linked visual representations.

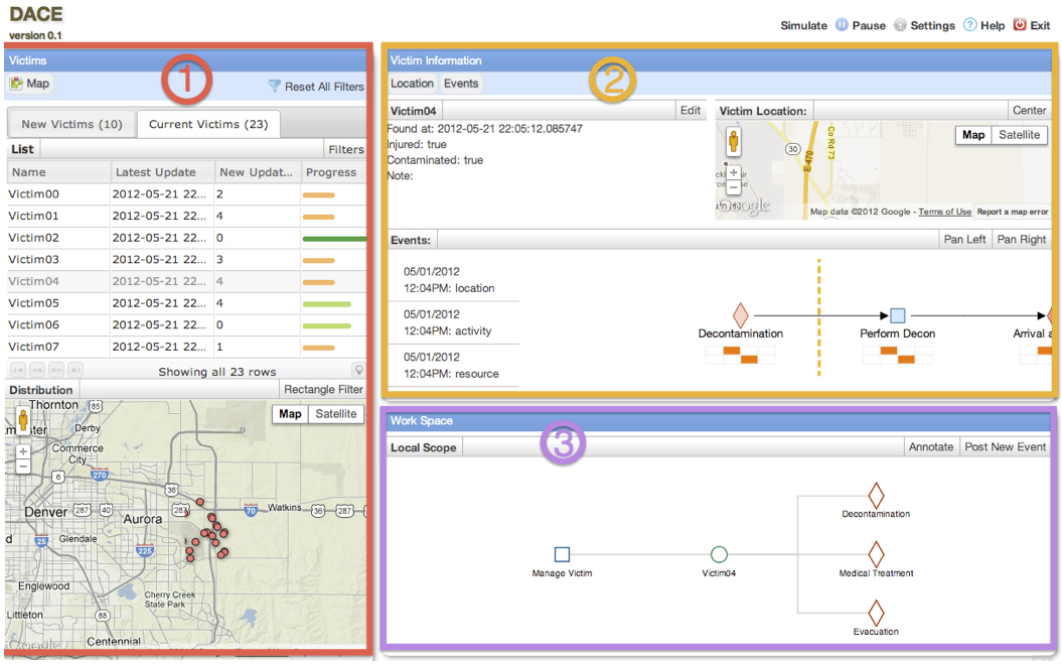


Figure 7.6. The main interface of the EDAP client

1. The *event view* provides a list of current events notified to the victim manager. The victim manager can click on each event to activate the event propagation tree to backtrack the event development.

2. The *activity view* shows the goals, activities, resources assigned to the current victim, which forms the local scope of the victim manager, where he/she can interpret the event and project state changes based on established understanding of the situation.

3. The *context view* consists of interactive data visualization tools to explore the contextual information. For the victim manager, the interface allows him/her to monitor the status of every victim that has been reported. A dynamic query interface is used for filtering the victims based on attributes, time, and geographical locations.

use the scenario to run through the whole process to

Chapter 8

Case Study: Promoting Awareness in Emergency Response

This chapter shows the feasibility and usefulness of the event-driven awareness promotion approach through a case study in the context of the emergency response. This case study aims to demonstrate the value of the event-driven awareness promotion approach in facilitating the emergency response operations by improved team awareness. The analysis of the case study consists in three steps:

1. The first step of the case study is to gather knowledge about the current status of awareness support in emergency response operations.

2. The second step includes the development of a concrete scenario, with the activities and events involved in the process. We describe the interaction between the activities and events, through which both the complexity and contingency scale up. On one hand is more events are generated as the activities are performed, and on the other hand is the increased number of adaptations that have to be made to the activities in order to address the problems caused by these events, such as resource assignment, top-down goal decomposition, and opportunistic re-planning.

3. The third step consists of the analysis of several use cases of the EDAP system in the scenario developed in the preceding step. It focuses on demonstrating the key strengths of our approach, as it would be if EDAP platform was used

to promote awareness in the scenario.

8.1 Awareness Support in Emergency Response

Emergency is any natural or man-caused situation that results in or may result in substantial harm to the population or damage to property [95]. Emergency response is the collaborative activity of gathering resources and acting upon the problems immediately after an emergency happens. While the scope of emergencies can be very broad (e.g. biological, chemical, nuclear/radiological, incendiary, or terrorist), the emergency operations share some common characteristics that make them fit into the scope of collaborative activities discussed in this study:

1. Emergency response is well recognized as a type of collaborative activities with a high level of complexity [113]. Emergency response usually involves multiple, distributed individuals and organizations to mitigate the impact of incidents that threaten public safety and the environment. During an emergency situation, workers rarely tackle tasks in an independent way. Instead, they find themselves working on a multitude of different activities, interleaving them in ways that seem best suited for getting their work accomplished given the practical pressures.

2. Furthermore, exceptions to the planned responses are a common and critical factor in these activities. An unplanned-for contingency may have its genesis in numerous circumstances [66]: an emergency situation may evolve, so that implemented plans are no longer applicable; it may be multi-faceted, requiring responding organizations to combine many plans in unexpected ways; and it may occur concurrently with other situations, thus creating resource shortages or outages.

With the high level of complexity and contingency in emergency response operations, supporting awareness has been recognized as an important component of emergency response management (ERM) systems [113]. Situation awareness is a critical element of emergency decision making in a variety of operational contexts. Existing studies have shown that improved situation awareness can benefit operational effectiveness by facilitating the planning process [60], improving the quality

and timeliness of decisions [13], and offering better feedback on the appropriateness of actions arising from such decisions [25]. With respect to supporting team-level awareness, research into and development of emergency response support has mainly focused on shared situation awareness [111], which has been considered as an essential prerequisite of an accurate collective picture of the emergency, and the basis of team decisions since they bring together and share individual perceptions, understandings, and predictions [60]. However, the distributed nature of team-level awareness, i.e. the fact that awareness should be considered as a partly shared and a partly distributed understanding of a situation among team members, has not been well recognized and supported in existing emergency response management systems [111].

Current computational environments for awareness support in emergency response management favor the event-based awareness models for two reasons:

1. The domain of emergency response has foundations in event driven task accomplishment [77]. Emergency response efforts are triggered by an incident, or a disaster, i.e. an "event" happening in the environment. To respond to the initial event, actors perform activities that add more events building up the situation. Decision makers have to react to these critical events for continuous response. Therefore the awareness they need is built up both from danger events as wells as emergency response events.

2. Actors in the emergency response operations have to focus on information needed and the immediate situation factors that relate to the decisions and actions they are taking [113]. As a result, they have very limited attentional resources to actively monitor the environment and each other's work in order to pick up the awareness information peripherally. The event-based model has the advantage in pushing the information to the actors so that they do not need to switch their attentions to the periphery until the event are notified to them.

Although much progress has been made to provide more effective event-based awareness support in emergency response operations with the integration of complex event-processing (CEP) engines (e.g. PRONTO [77] and PLAY projects

[112]), the limitations of event-based models as we discussed in Section 3.2 also exist in the emergency response domain:

1. The effectiveness of event-based models largely depends on the quality of event subscriptions, which often requires considerable effort from the actors to explicitly manage. With the high level of complexity and contingency in emergency situations, existing event notification mechanisms either lead to too many irrelevant events notified to the actors, or the actors have to frequently modify subscriptions to adapt to their changing needs.

2. The current event-based systems in the emergency response domain focus on the external events that are generated by physical sensors or complex events that are calculated by the rule-based event processing agent [77], little support has been given to the social development of awareness, i.e. the internal events generated by the human actors.

In the following of this Chapter, we describe a concrete emergency response scenario and then use it to argue how these limitations can be addressed in our event-driven awareness promotion approach.

8.2 An Emergency Response Scenario

For this case study, we consider a nuclear crisis scenario in which a large quantity of radioactive substance is accidentally released in the atmosphere, due to a critical accident in a nuclear plant. The multiplicity and diversity of actors involved, the volume and heterogeneity of information, the critical dependencies between actions as well as the dynamics of the environment make the situation more complex.

This scenario is based on the document analysis from several government websites, including the National Response Framework, the Nuclear/Radiological Incident Annex to NRF, NRC Incident Response Plan, as well as a collection of FEMA after action reports related to specific emergency exercise. To simplify the analysis, however, we reduce the complexity of the emergency response operations significantly in this scenario, comparing with the real world situations described in these documents.

8.2.1 Overview of the scenario

8.2.1.1 The field of work

Source of the incident The radiation leak in this scenario originates from the combination of two problems in a nuclear plan:

1. Due to the wearing effect of time, a leak appeared in the steam generator. As a result, the water within the primary loop connecting the reactor and the steam generator, contaminated, spreads through the secondary loop.

2. The throttle valve, a safety device of the secondary loop, opens due to the increased pressure inside the secondary loop. It does not respond to the manual bypass of the safety loop. As a result, the steam of the secondary loop, contaminated, escapes to the atmosphere.

Actors To respond to the emergency caused by the radiation leak, many stakeholders are involved. The emergency operation center (EOC), in charge of operation, is formed with federal agencies, state and local public safety officials. Besides, firemen, policemen, and any other actors involved in the response process has one representative at the emergency operation center, to validate the feasibility of decisions, link with the field and ensure communication between actors. The EOC is distributed. Most of the decisions are made locally, where delegates are gathered, but decisions may also come from the national authority, state or local officials, or experts.

Besides the EOC, a large number of actors work in the filed to operate or support the nuclear crisis response. This includes (1) firemen to distribute iodine and rescue victims from the impacted area, (2) police to confine population and guide the evacuation, (3) decontamination and medical teams to assist victims, (4) transportation team to provide prompt delivery of personnel, equipment, and victims, (5) and scientific team to survey radiation, provide meteorologic information, and assess the evolution of the situation.

Resources A variety of resources are used by the actors in the scenario to perform their tasks. Some of the resources are used to perform specific tasks, such as the measuring equipment used by the scientific team to survey radiation, the

medical equipment used by the decontamination and medical teams to assist victims. On the other hand, some of the resources are shared by multiple actions. The rescue vehicles can be used for transporting victims, delivering actors or other resources. The victims are manipulated by multiple actors in different tasks, such as decontamination, medical treatment, or transportation. The mobile decontamination or medical stations are used for operations on multiple victims.

Actions The emergency response in the scenario is consisted of a large number of actions. We follow the analysis in [112] to structure the actions into three kinds of processes: decisional, operational and support process.

1. The first process is dedicated to present the decisional part of the response operation. It concerns decision taking during the emergency response to plan and control the operational and support processes. These decisions help nuclear plant teams and public services to control nuclear repairing, to mobilize protection and relief resources, to communicate with media and local authorities and to animate the actual actions performed in the field.

2. The operational process aims to resolve nuclear accident and its consequences. It concerns the actual actions performed in the field, such as protecting populations and rescue employees and populations.

3. The support process concerns the supporting actions dedicated to provide means to other processes. This consists of (1) making available resources and means and (2) assessing the situation continuously.

Table 8.1 shows a summary of the higher level structure of the actions. For each of these actions, subsidiary actions can be identified. These subsidiary actions can be further decomposed into more detailed actions. It is possible that each action can be decomposed in several ways following different plans. For example, the medical treatment on a victim can be performed by sending the victim to one of the medical stations and then treat the victim at the station, or if the victim is in an urgent situation, a medical team needs to be sent to the victim for immediate treatment.

Type	**Action**	**Subsidiary actions**	**Invovled actors**
Decisional	To plan and control the operational and support processes	1. To assign actors and relief resources 2. To communicate with media and local authorities 3. To activate operational actions	Federal agency, Local officials, Delegates from different departments
Operational	To protect population	1. To alert/communicate 2. To confine population 3. To distribute iodine capsules 4. To evacuate population	Firemen, Police, Transportation team
Operational	To rescue victims	1. To search for victims 2. To decontaminate 3. To treat physical injuries 4. To transfer to new accommodation or hospital	Firemen, Decontamination team, Medical team, Transportation team
Support	To support all reponse operations	1. To deliver victims 2. To find and deliver equipment 3. To control traffic	Transportation team, Police
Support	To assess situation	1. To measure radioativity 2. To measure weather characteristics 3. To provide situational reports continuously	Scientific team

Table 8.1: Actions in the nuclear emergency response scenario

8.2.1.2 Events in the scenario

The demand for knowledge about events arises within the different processes in the scenario. Based on the sources where they come from, the events in the scenario can be clarified into five categories:

1. *"Incident"* events represent actual or possible changes of the emergency incident. This type of events are generated by the scientific team during the

action of assessing the emergency incident. Examples of these events include the changes to the measurements taken from the field, such as the change to the level of radiations, wind velocity and direction, as well as the results of analysis based on these measurements, such as the increased impacted area of the incident.

2. *"Risk"* events refer to the noticeable environmental changes that lead to (or potentially lead to) one or several risks on the emergency response operations. It may be linked to transportation risks as an accident or a traffic jam occurs. Or It may be the change of weather situations (e.g. raining) or fire/explosions risks that can complicate the response operations.

3. *"Resource"* events refer to the changes of resources, i.e. change of their availability in time and space, change of attribute values (e.g. quantity), or assignment to actions.

4. *"Actor"* events refer to the changes of actors, such as the change of their locations, roles, or capabilities.

5. *"Action"* events refer to the changes of actions, such as the changing execution state (e.g. the initiation, completion, or delaying of an action), the condition satisfaction, or the change of plan.

Table 8.2 lists some examples of the events in the scenario with relation to associated event types and possible event sources that produce them.

Example Events	**Event Type**	**Event Producer**
Incident event: Radioactivity level within 5 mile of the plant exceeded 50 mSv/h	Vale comparison	Scientific team to measure radioativity
Risk event: A traffic jam occurs on the road between A and B	Existential change over time	Transportation team to deliver victims
Actor event: The army, who is not supposed to be involved in iodine distribution, has to take part in the process	Conceptual relation	Officials to plan and control the operational actions

Resources event: Electricity generators will arrive at the decontamination station in 1 hours...	Spatio-temporal relation	Transportation team to deliver the equipment
Action event: Iodine capsules have to be distributed to people within 5 mile of the plan	Existential change over time	Officials to plan and control the operational actions
Action event: New road signs have been placed to support the evacuation plan	State change	Police to control traffic

Table 8.2: Events in the nuclear emergency response scenario

8.2.2 Event-driven development of the field of work

An important characteristic of the complex collaboration as this emergency response scenario is about to demonstrates is the interaction between the field of work and events. In the beginning, the aspects of the field of work in emergency response are changed by some initial events. Upon receiving these events, actors interpret them to assess the situation, then make decisions or perform actions that lead to more changes to the current field of work. These new changes can generate new events that trigger more changes to the filed of work. In this way, the collaborative activity is developed.

With the various entities in the field of work and the large number of events that can be generated in the scenario, it is a challenging task to describe the development of the whole emergency response process. To simplify the analysis of the scenario in the next section, we focus on the development of a sub-process of the scenario, i.e. the action to rescue victims from the impacted area. This sub-process itself is driven by a series of events and also generates new events. We consider several consecutive episodes in the process of rescuing a victim. Each of the episodes represents an example of a typical development trajectory of the collaborative activity that serves as the basic unit of analysis in this case study.

Table 8.3 lists the four development trajectories that may happen in collabo-

rative activities, and in the following of this section we describe the corresponding episodes representing them in the process of rescuing a victim.

Trajectory Type	**Description**	**Episode**
Goal activation	A goal is activated by some triggering event, and then the action to achieve the goal is developed through top-down decomposition	Episode 1
Plan development	An event causes a new problem that is not addressed by the current plan of an action. In order to the fix the problem the plan has to be further developed.	Episode 2
Responsibility transfer	An event leads to a breakdown in performing an action. To fix the problem, some actor has to perform an action that is out of the pre-defined responsibility.	Episode 3
Opportunistic re-planning	An event leads to the decision that the current plan of an action has to be abandoned and a new plan is developed.	Episode 4

Table 8.3: Development trajectories in the field of work

8.2.2.1 Episode 1: Goal activation

The first episode represents the initial development of the rescue action triggered by an event (Figure 8.1).

1. It starts with the fireman's action to search for victims in the impacted area. When the fireman finds a new victim, she reports the discovery of the new victim as a new event.

2. The event about the new victim is received by the victim manager (VM) and activates the VM's goal that the victim needs to be rescued. The VM decomposes the action to rescue the victim into sub-actions, and first evaluates the status of the victim.

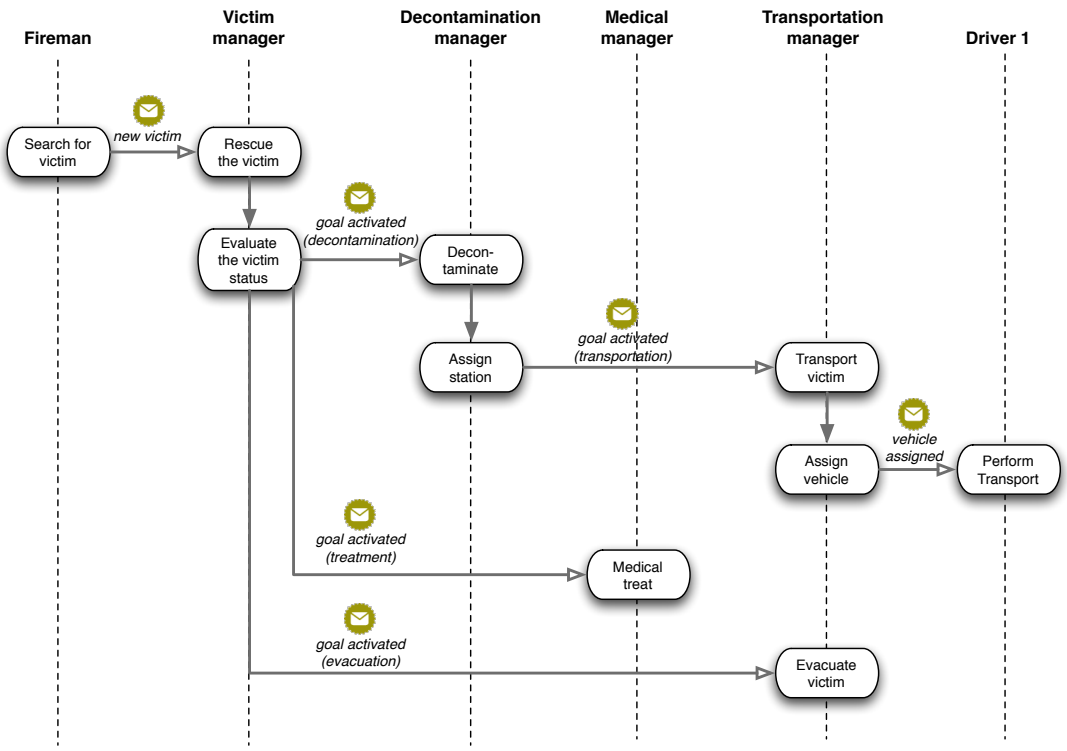


Figure 8.1. Episode 1: Goal activation

3. Because the radiation level of the victim exceeds the threshold, the victim needs to be first decontaminated. As a result, the VM generates a new event indicating the goal to decontaminate the victim is activated.

4. Because the victim is also physically injured, the victim needs to be treated at a medical station after decontamination. As a result, the VM generates another event indicating the goal to treat the victim is activated.

5. After the treatment, the victim also needs to be evacuated to a shelter for recovery. The third event generated by the VM describe the goal to evacuate the victim is activated.

6. Upon receiving the event to activate decontamination, the decontamination manager DM infers that the action to decontaminate is within her responsibility and she is capable of performing it, so she is committed to performing the action, decompose the action into sub-actions.

7. To perform decontamination, DM first needs to assign a decontamination station to the victim. After the assignment of station ($DS1$) to the victim, the DM activates the goal to transport the victim to the station, which is described as a new event.

8. Upon receiving the event that activates the goal to transport the victim to the station, the transportation manager TM infers that the action to transport the victim to the station is within her responsibility and she is capable of performing it, so she is committed to performing the action, and further decompose the action.

9. TM first assigns a driver ($DR1$) with a rescue vehicle to the action, and an event describing the assignment of the driver ($DR1$) to the action to transport the victim is generated. Because the actual transportation action is out of the DMs responsibility, she is waiting for the driver to perform it.

10. Then the driver $DR1$ receives the event about the assignment and starts to perform the action.

11. Besides, the medical manager MM receives the event to activate the goal to treat the victim. However, because this action depends on the action to decontaminate the victim to be completed first, she has to wait for now.

Episode 1 shows an example of how an initial event can trigger the actor to activate the goal to perform an action, and lead to top-down decomposition of the action. During the development of the action, more events are generated and more actors are participated into the action. The complete development trajectory of Episode 1 can be found in the Appendix (A.1).

8.2.2.2 Episode 2: Plan development

Episode 2 is built on top of Episode 1, showing how the current plan of the rescue action is further developed due to a problem caused by an event (Figure 8.2).

1. Episode 2 starts with an event that is reported by an operator at $DS1$, where the decontamination of the victim will be performed. The event shows that

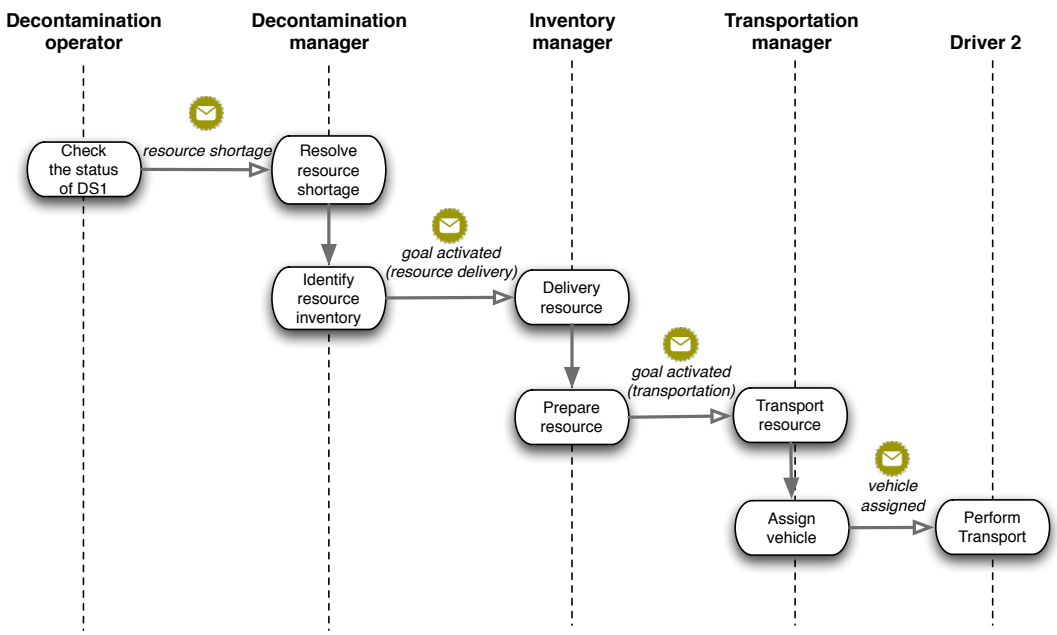


Figure 8.2. Episode 2: Plan development

the quantity of an important resource, i.e. the available rinse tanks in $DS1$ is running low.

2. The DM receives this event and infers that the $DS1$ cannot work properly due to this resource shortage. In order to ensure the victim is successfully decontaminated, a new action to resolve the resource shortage needs to be added to the current plan of rescuing the victim.

3. The DM first needs to locate an inventory where the rinse tanks can be supplied. After identification of the inventory, the DM activates the goal to deliver the resource from the inventory to the station, and a corresponding event is generated.

4. Upon receiving the event about activating the goal to deliver the resource, the inventory manager IM starts a new action to deliver the resource. The IM first needs to prepare the resource for delivery. Once the resource is ready, she activates the goal to transport the resource to the station $DS1$, and the corresponding event is sent to the TM.

5. Upon receiving the event, TM starts a new action to transport the resource.

TM first needs to assign a driver $DR2$ with a rescue vehicle to the action, and an event describing the assignment of the driver ($DR2$) to the action to transport the resource is generated.

6. Then the driver $DR2$ receives the event about the assignment and starts to perform the action.

Episode 2 shows an example of the interleaving of planning and acting. Although this additional action (i.e. the action to resolve resource shortage) is not a required subsidiary component in the initial plan of performing the higher-level action (i.e. to rescue the victim), it is later added to the plan because of the problem of a resource is identified during the performance of the action. Details of Episode 2 is described in the Appendix (A.2).

8.2.2.3 Episode 3: Responsibility transfer

Episode 3 starts with an event leading to a breakdown in performing a subsidiary action, which can cause problem on the whole rescue action. To fix the problem, some actor has to perform an action that is out of her pre-defined responsibility (Figure 8.3).

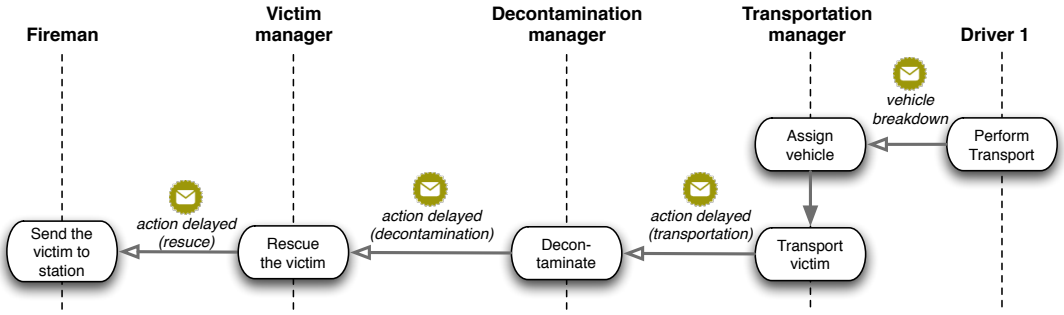


Figure 8.3. Episode 3: Responsibility transfer

1. In the beginning of Episode 3, the rescue vehicle with the driver $DR1$ encounters a mechanical breakdown, as the driver is on the way to pick up the victim. The driver reports the problem as the initial event of Episode 3.

2. The TM receives the event and infers that because of the vehicle breakdown, the $DR1$s action to transport the victim cannot be achieved. To solve the problem, the TM attempts to assign another driver to perform the action. However, because all the drivers are currently in duty, the TM cannot find a driver to perform the action until a later time. This problem is expressed as an event showing the action to transport the victim has to be delayed because of the unavailability of drivers.

3. Upon receiving the event generated by TM, the DM further infers that because of the delay to transport the victim, the action to decontaminate the victim will also be delayed, and a corresponding event is generated.

4. Similarly, the VM infers that because of the delay to transport the victim, the action to rescue the victim is delayed.

5. The event generated by VM is then received by the fireman who is with the victim in field. The fireman infers that although the delivery of the victim is not usually in the scope of her responsibility, she can help the rescue action by sending the victim directly to the decontamination station $DS1$. As a result, the fireman activates the goal to send the victim to $DS1$.

6. Upon receiving the event from the fireman, TM infers that because of the new action of the fireman, the responsibility of transporting the victim is now transferred from her to the fireman.

Episode 3 demonstrates two important natures in the development of collaborative activities. First, it shows how the effect of one action can cascade to the other actions because of the dependencies among them. Second, it shows the transferability of actors' responsibilities in a complex collaboration. In order to achieve the shared goal of the group, the actors may have to undertake tasks that are not specified in their roles. Details of Episode 3 is described in the Appendix (A.3).

8.2.2.4 Episode 4: Opportunistic re-planning

The last episode shows the most significant change to the existing plan to rescue the victim. Because of the status change of the victim, the current plan to rescue

the victim is no longer applicable and has to be replaced with a new plan (Figure 8.4).

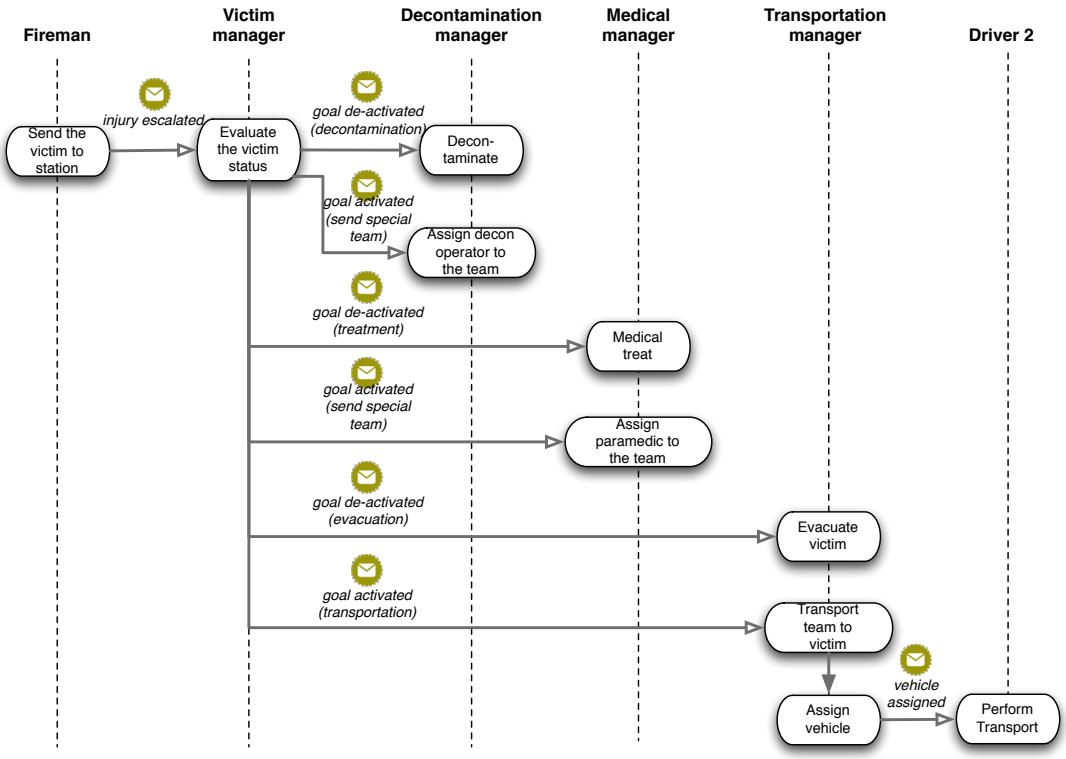


Figure 8.4. Episode 4: Opportunistic re-planning

1. As the fireman decides to directly send the victim to the decontamination station, she finds that the physical injury of the victim becomes severe and prevents the fireman from moving the victim into the vehicle. The fireman has to report the status change of the victim as a new event.

2. The event is received by the VM. VM evaluates the situation of the victim and decides that because of the serious injury, the original plan to rescue the victim is no longer appropriate. A new plan needs to be adopted: a special team needs to be dispatched to the victims location to decontaminate and treat the victim on spot, and then the victim is directly sent to the hospital for further treatment. The changes to the plan are represented as a set of events and distributed to relevant actors.

3. Upon receiving events from the VM, DM first removes the current action to decontaminate the victim from her responsibility. Then DM infers that in order to perform the action to dispatch a special team to the victim, a decontamination operator needs to be assigned to the team. By reviewing the available operators, DM decides on the operator who will be sent to the victim, and generate an event to indicate the assignment of the operator to the team.

4. Similar to DM, the MM first removes the action to treat the victim from her current responsibility, and decides on the paramedic that will be assigned to the special team. An event to indicate the assignment is then generated.

5. The TM removes the action to evacuate the victim from her current responsibility. Then the TM infers that in order to perform the action to dispatch a special team to the victim, the action to transport the team to the victim needs to be activated. TM needs to assign a driver with a rescue vehicle to the action. Because the deactivation of the original decontamination action, the driver $DR2$ who was assigned for delivering the equipment during the decontamination action is now available for this action. As a result, TM generates the event to describe the assignment of $DR2$ to the action to transport the special team to the victim.

6. The driver $DR2$ receives the event about the new assignment and starts to perform the action.

Episode 4 shows how one action can be performed using different plans in different situations. Triggered by the initial event, the action has to be opportunistically re-planned to adapt to the changing environment. In the re-planning process, the resources that are bound to the old plan are released and can be assigned to other actions in the new plan. For example, the rescue vehicle with the driver $DR2$ was assigned to perform the resource delivery action, however after the re-planning, it is now re-assigned to a new action to deliver the rescue team (Details of Episode 4 is described in Appendix (A.4)).

8.2.2.5 Discussion

The four episodes demonstrates several characteristics of the collaborative activity in the emergency response scenario:

1. From the four episodes, we can see how the complexity of the collaborative activity is coupled with the different types of contingency. In the beginning, the activity can be quite simple, with limited actors involved and standard work flow. However, due to the various exceptions that may occur in the process, much more effort is given to fix the problems and breakdowns in the collaborative activity. As a result, more actors are involved into the field of work, and the plan of the activity has to be modified significantly. The coupling of complexity and contingency makes the actors' awareness need become unpredictable in the beginning of the collaborative activity.

2. These episodes show good examples of the interaction between events and field of work. On one hand is the events triggering the development of the field of work, and on the other hand is the changes to the field of work generate new events. It is within this recursive cycle between the events and the field of work that the knowledge about the collaborative activity is built up. This provides the basic for our approach and in next section, we will demonstrate how the knowledge updating process in our approach works in this scenario.

3. These episodes also illustrate the social development of awareness across multiple actors. Along with the development of the activity is how the events are generated by different actors to express their interpretation of the situation, and then are consumed by other actors. In Section 8.4, we will demonstrate how the social development of awareness in these episodes can be supported in our approach.

8.3 Knowledge representation and updating in the scenario

The goal of this section is to examine the expressiveness of the PlanGraph model in our approach to represent the field of work and validate the knowledge updating process within the context of the scenario. To achieve this goal, we simulate the development of the four episodes described in previous section in our prototype system, the EDAP platform. We walk through the four episodes to investigate how the PlanGraph is updated by the *Knowledge Updating Module*, driven by the set of events occur in the episodes. Later, the simulation results allow us to analyze the characteristics of the field of work in the scenario.

In general, the simulation is conducted in three steps:

1. First, we engineer the knowledge base to model knowledge about all the actions, recipes, and actors that are described in the four episodes. For example, Figure 8.5 shows the two recipes associated with the action to rescue the victim that have been described in Episode 4.

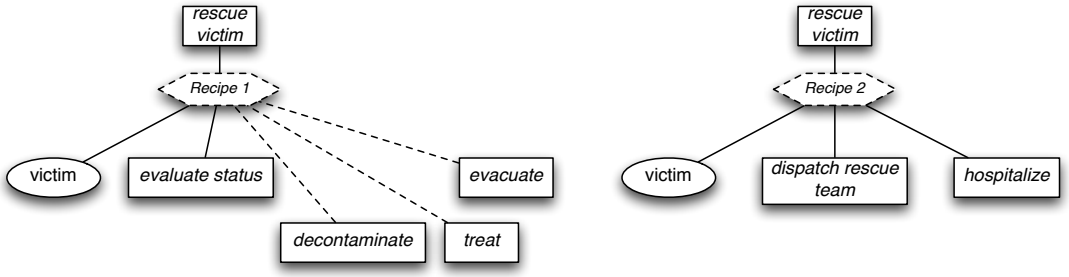


Figure 8.5. Example recipes in the scenario

2. Then, we develop an event simulator to input the events that occur in the episodes to the EDAP server. The event simulator is a simple command line tool that reads a pre-defined script recording the sequence of events that are described in the four episodes. At a given time interval, the event simulator emits one event from the sequence and sends it to the *Event Publishing Service* in the EDAP server via HTTP request.

3. As we described in Section 7.2.1, each of the events received by the *Event Publishing Service* is then passed to the *Knowledge Updating Module* to update the PlanGraph model. We persist the PlanGraph during each step in the database for analysis.

In the following of this section, we report the simulation results including the development of the PlanGraph model and the constructed local scopes and dependencies in the four episodes. The former is to show the capability of the system to keep track of the event-driven development of the field of work in the episodes, and the latter is to analyze the three characteristics of the field of work as we conceptualize in Section 2.4.2.

8.3.1 Simulating the knowledge updating

8.3.1.1 Episode 1

In Episode 1, the first event sent to the system is an event reporting that a new victim has been found by the fireman ($E1$). The event has an event type *'NewVictim'* and has several key attributes (e.g. id, detection time, name, location, radiation level, injury information, etc.).

In response to this event, the *Knowledge Updating Module* performs the four-step knowledge updating process.

1. First, the system attempts to associate with any existing entities in the PlanGraph model. Because this is the start of the activity, the PlanGraph is empty at the moment and no association can be found.

2. Second, the system checks whether the event can lead to changes towards the action performance or trigger new action in the assessment step. In our knowledge base, we store an association rule that indicates that every *'NewVictim'* will activate the action to rescue it. Following this activation rule, a new action node *'rescue victim'* is added to the PlanGraph as the root node representing this new action to rescue the victim.

3. After the assessment, the system finds that a new action has been added to the PlanGraph, which triggers the elaboration process. The system searches

the knowledge base to find a recipe for the *'rescue victim'* action. In our knowledge base, we have two recipes that are associated with the *'rescue victim'* action and the system chooses the first one that represents the normal work flow to construct the default plan of the action. During the elaboration process, the *'victim'* parameter is assigned with the value stored in the event. The *'evacuate status'* action is added to the plan. Because the rest of the subsidiary actions (*'decontaminate'*, *'treat'*, *'evacuate'*) are optional, i.e. whether they should be added to the plan depends on the result of *'evacuate status'* action, the recipe selection stops. In addition to the recipe selection and parameter binding, the elaboration step also involves searching for the actors who might be potentially intended to or capable of performing the action. By retrieving this knowledge from the knowledge base, the system believes that the victim manager VM is potentially intended to perform the action and have the capability to perform the *'evacuate status'* action. This is used to update the corresponding *'Intentions'* and *'Capabilities'* attributes attached with the PlanGraph nodes.

4. Because the event does not indicate any state change on current actions, the propagation step is skipped.

Figure 8.6 shows the PlanGraph after the knowledge updating process on the first event ($E1$) in Episode 1.

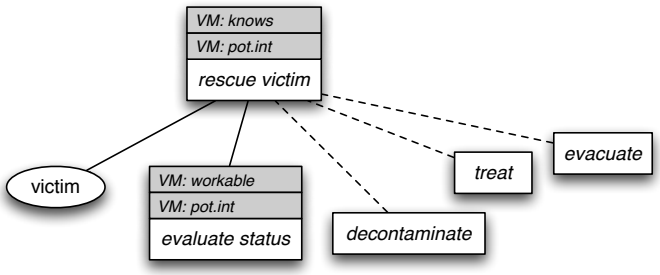


Figure 8.6. PlanGraph after $E1$ in Episode 1

The second event ($E2$) in Episode 1 is an internal event generated by the victim manager VM to indicate the intention that ($int.th$) the *'decontaminate'* action is performed, based on the result of the *'evacuate status'* action. This intention event

has several key attributes, including the actors id (VM), the action he intends to perform (*'decontaminate'*), and the intention type ($int.th$).

1. As the system receives this event $E2$, the system first attempts to associate it with any existing entities in the PlanGraph model. Because this is an intention event, the system traverses through the PlanGraph to search for any match between the *'decontaminate'* action mentioned in the event and the action nodes in the PlanGraph. When the system is able to find such a match, the system uses the information in the event to update VMs intention level on it.

2. During the assessment step, the system could not find any applicable association rules.

3. Because the *'decontaminate'* action is now added to the PlanGraph as a new action, the system searches the knowledge base to find a recipe for the *'decontaminate'* action, and use it to elaborate the PlanGraph. Because the system does not have knowledge about how to identify the parameter *'decontamination station'* of the *'decontaminate'* action, the PlanGraph cannot be further elaborated. Besides, the system believes that the decontamination manager DM has potentially intention to perform the *'decontaminate'* action and knows how to perform the action. This is used to update the corresponding *'Inentions'* and *'Capabilities'* attributes attached with the PlanGraph nodes.

4. Because the event does not indicate any state change on current actions, the propagation step is skipped.

The $E3$ and $E4$ are very similar to $E2$, as they are also internal events generated by the victim manager VM to indicate the intention that ($int.th$) the *'treat'* and the *'evacuate'* action is performed, based on the result of the *'evacuate status'* action. We skip the details here, and Figure 8.7 shows the PlanGraph after these events are processed.

Events $E5$, $E6$ are generated by the decontamination manager DM in the process of elaborating the action to decontaminate the victim. The system processes these events and elaborate the PlanGraph model in the similar way as the previous

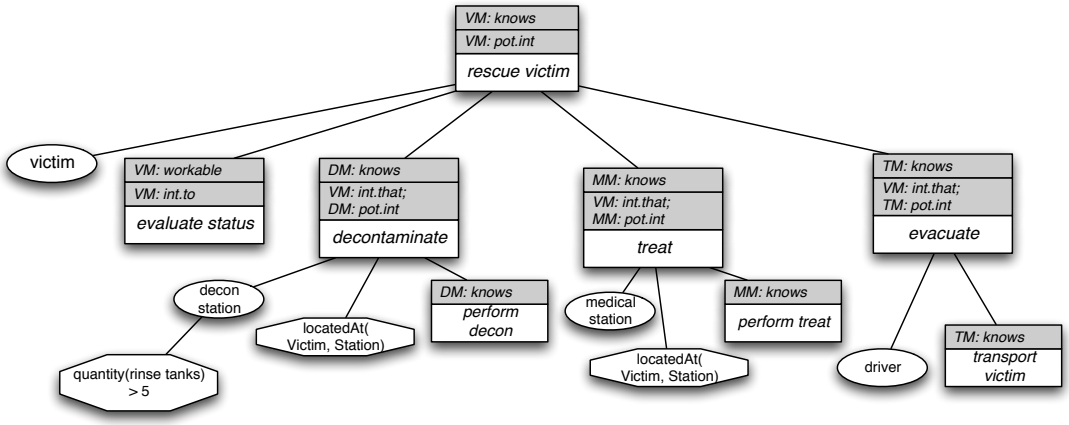


Figure 8.7. PlanGraph after $E4$ in Episode 1

events. Finally, the event $E7$ is generated by the transportation manager TM to further elaborate the action to transport the victim to the decontamination station. We skip details about these events as well since the system's reasoning processes on them are very similar to previous ones. Figure 8.8 shows the PlanGraph after all the events in Episode 1 are processed.

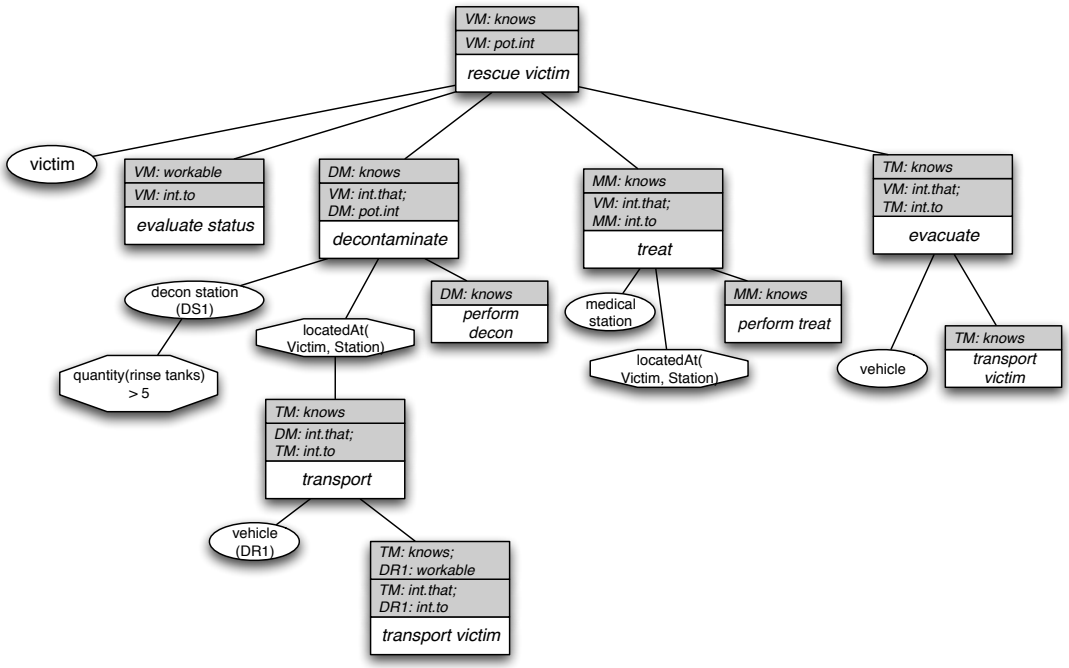


Figure 8.8. PlanGraph after Episode 1

8.3.1.2 Episode 2

In Episode 2, the first event sent to the system ($E1$) is a resource event reported by an operator at $DS1$, where the decontamination of the victim will be performed. The event shows that the quantity of available rinse tanks in DS1 is running low. The four-step knowledge updating process that is triggered by this event is summarized as follow:

1. In the beginning, the system associates the event with the *'decon station'* parameter under the *'decontaminate'* action, as it indicates the value change of an attribute (i.e. quantity of rinse tanks) of the station, and updates the corresponding values of the parameter.

2. Second, the system checks whether the event can lead to state changes of the entities in the PlanGraph model. By evaluating the conditions associated with the parameter, the system finds that the condition that the quantity of the rinse tanks should be above 5 can no longer be satisfied because of this event. As a result, a new derived event is generated by the system to indicate the state change of this condition (from *'holding'* to *'open'*).

3. During the elaboration step, since the condition is no longer holding, the system searches the knowledge base to find any action that can be performed to satisfy the condition. In our knowledge base, we have an action *'resolve resource shortage'* that can be used to satisfy the condition, and therefore the system add the action to the PlanGraph and load the recipe to perform the action. The system applies the initial intention *pot.int* and capability *knows* of the DM to this new action. Because the system does not have knowledge about how to identify the parameter *'inventory'*, the PlanGraph cannot be further elaborated.

4. Because a state change event has been derived in the assessment step, the system attempts to predict future state changes in the field of work in the propagation step. A Bayesian network is constructed based on the current PlanGraph model and used to reason how likely the other entities in the PlanGraph will be impacted by the initial state change. After the Bayesian

network-based reasoning, the system finds that the state change of this condition (from *'holding'* to *'open'*) will lead to the failing of the *'decon station'* parameter, which may be further propagated to the upper-level *'decontaminate'* and *'rescue'* actions.

Figure 8.9 shows the PlanGraph after the knowledge updating process on the first event ($E1$) in Episode 2, and Figure 8.10 shows the event chain generated by the system in the process.

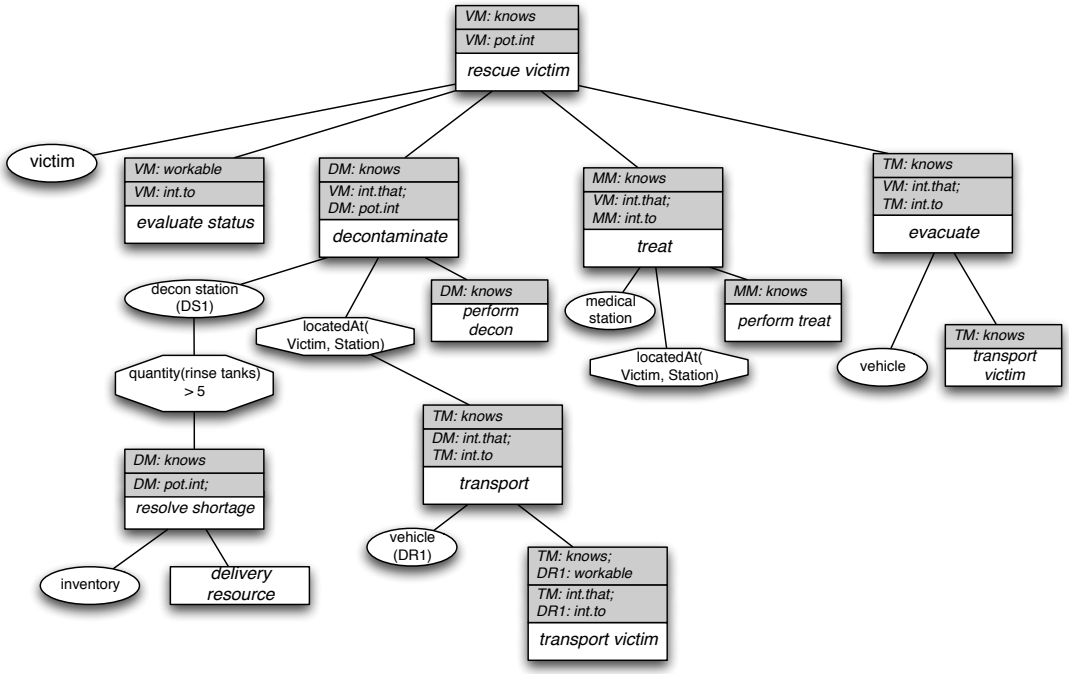


Figure 8.9. PlanGraph after $E1$ in Episode 2

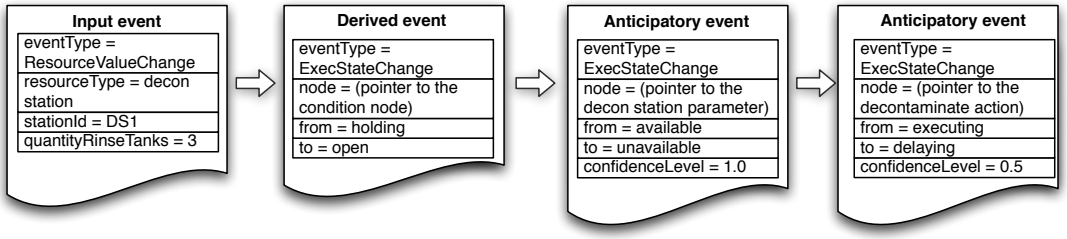


Figure 8.10. Event chain after $E1$ in Episode 2

The following steps in Episode 2 are developed in the same way as the goal activation process in Episode 1. The decontamination manager DM identifies the *'inventory'* parameter, and then activates the goal to deliver the resource from the inventory to the station $DS1$. The inventory manager then works on the *'deliver resource'* action by preparing the resource and activates the goal to transport the resource to the station. The transportation manager then further elaborate the transportation action. Figure 8.11 shows the PlanGraph after all the events in Episode 2 are processed.

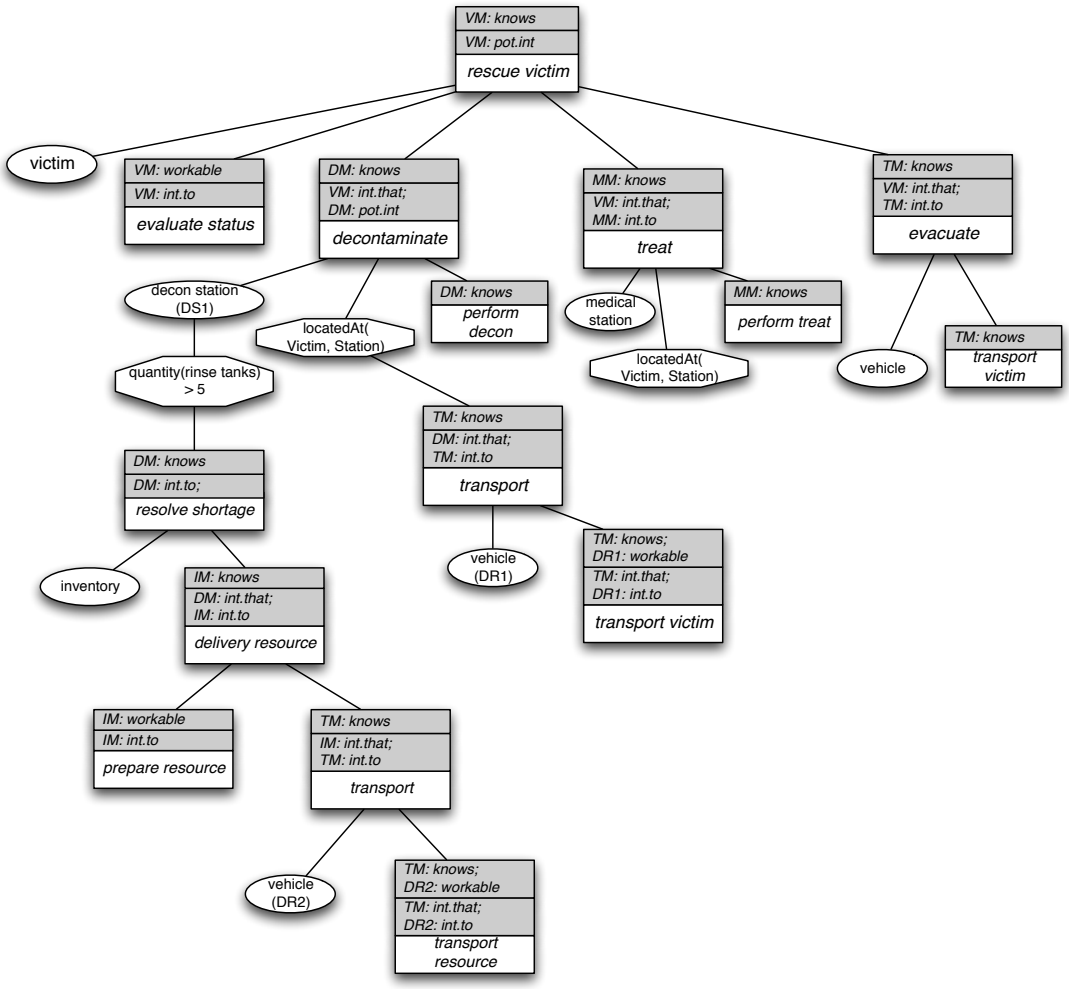


Figure 8.11. PlanGraph after Episode 2

8.3.1.3 Episode 3

Episode 3 happens after driver dr_1 starts the action *'transport victim'* and is on the way to pick up the victim. It is an external event indicating a status change of the rescue vehicle with driver dr_1, i.e. it encounters a mechanical breakdown. The event has an event type *'ResourceStatusChange'* and carries information about the driver, the vehicle, and the current status of the vehicle. The knowledge updating process performed on this event by the system is described as follow.

1. When the event is sent to the system, the system attempts to associate with any existing entities in the PlanGraph model. Because this is an external event indicating an attribute change on a resource, the system traverses through the PlanGraph to search for any match between the resource (*'vehicle'*) mentioned in the event and the nodes in the PlanGraph. When the system is able to find such a match, the system uses the information in the event to update the value attached to the parameter (*'vehicle'*).

2. Then the event is further processed in the assessment step. A simple assessment rule is applicable in this case, i.e. if the status of the rescue vehicle is changed to breakdown, the parameter that is assigned with this resource becomes unavailable. As a result, a new state change event *'ExecStatChange'* on the execution state of the parameter is derived.

3. As nothing can be done by the system to fix this problem by searching the knowledge base, the elaboration is done with nothing added to the PlanGraph.

4. Because a state change event has been derived in the assessment step, the system attempts to predict future state changes in the field of work in the propagation step. After the Bayesian network-based reasoning, the system finds that the execution state of the dr_1's action to transport the victim (*'transport victim'*) is changed from *executing* to *failing*, and a new anticipatory event describing this state change is generated, and added to the output event chain. Furthermore, the state change is propagated to the higher level *'transport'* action with a confidence level of 0.5, as the system infers that the

action is likely to be impacted, meanwhile the problem can be resolved by assigning another vehicle to perform the *'transport victim'* action.

As we can see in the knowledge updating process, the initial event is augmented into an event chain with four events (Figure 8.12): the original external event $E1$, the derived event describing the execution state change of the parameter *'vehicle'*, and the two anticipatory event predicting the execution state change event on the action *'transport victim'* and the *'transport'* action. The processing of $E1$ in Episode 1 shows the capability of the system to enrich the original event by performing reasoning on the PlanGraph.

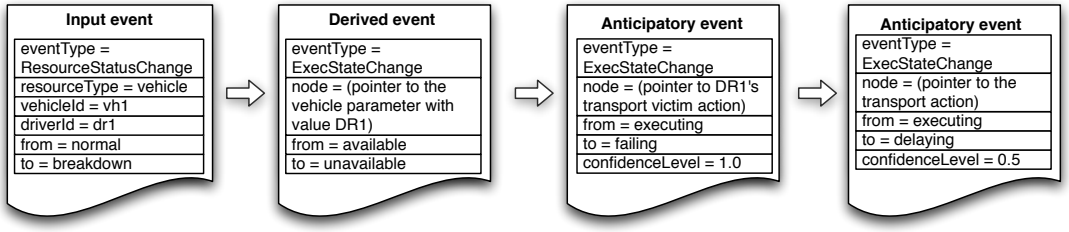


Figure 8.12. Event chain after $E1$ in Episode 3

The following events $E2$, $E3$, and $E4$ are the internal events that are contributed by the transportation manager (TM), decontamination manager (DM), and victim manager (VM) respectively. These events are used to express the corresponding actors' interpretation on the initial event $E1$. As receiving $E1$, the TM first attempts to fix the problem by assigning another driver to the *'transport victim'* action. However, because all the drivers are currently in duty, the TM cannot find a driver to perform the action until a later time. As a result, the TM generates $E2$ to confirm the state change of the *'transport'* action that was previously inferred by the system. This new event $E2$ is then received by the DM, and then DM further derives another event $E3$ to express her belief that the *'decontaminate'* will also be delayed because of $E2$. In the similar way, VM generates the other event $E4$ to propagate the state change.

The processing of these events allow the system to update the event chain starting with $E1$ using human actors' interpretation, and in this way the event propagation tree can be generated. Figure 8.13 shows the active event propagation chain that is derived from the event propagation tree, spanning from $E1$ to $E4$.

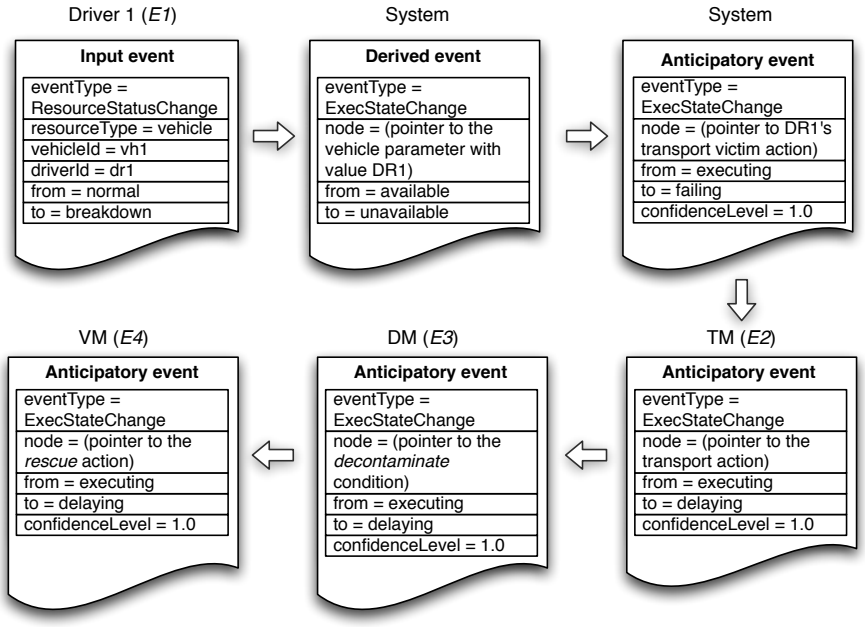


Figure 8.13. Active event propagation chain after $E4$ in Episode 3

The last event $E5$ in Episode 3 indicates the fireman's reaction to the situation developed through $E1$ to $E4$. Although delivery of the victim is not in the scope of the fireman's responsibility, she can help fix the problem by directly sending the victim to the decontamination station $DS1$. By receiving $E5$, the system updates the PlanGraph by revising the plan of the *'transport'* action: (1) first the fireman's intention to perform the *'transport'* action is updated in the *Intentions* attribute of the action node during the association step, (2) the value of the *'vehicle'* parameter is now replaced by the vehicle assigned to the fireman, (3) and then the fireman's action to send the victim, i.e. *'send victim'* is added to the *'transport'* action as a sub-action. Figure 8.14 shows the PlanGraph after all the events in Episode 3 are processed.

8.3.1.4 Episode 4

Episode 4 starts with an event $E1$ that indicates the attribute change of a victim, i.e. the injury information is changed. The system associates this event with the *'victim'* parameter under the *'rescue'* action, and update the attribute storing the injury information. However, such descriptive information cannot be directly

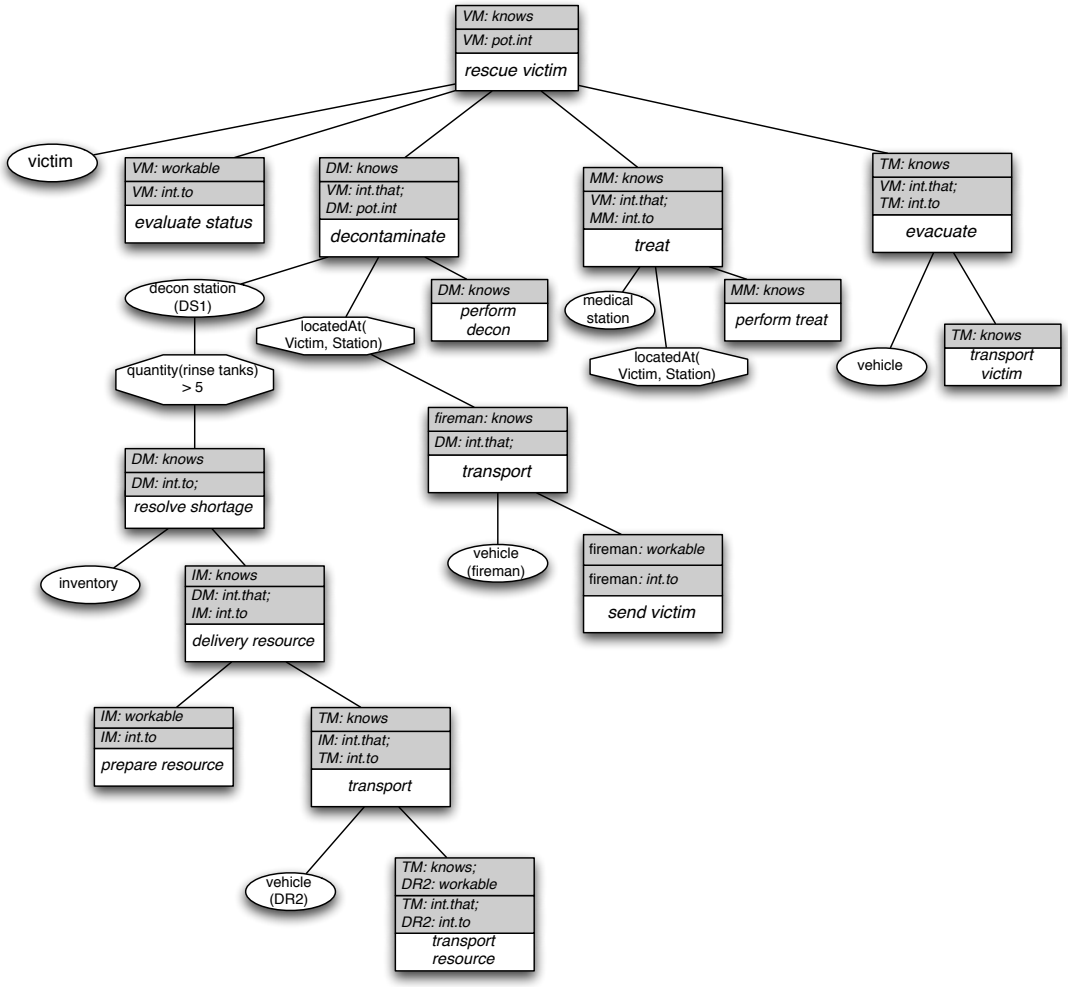


Figure 8.14. PlanGraph after Episode 3

interpreted by the system, all the following three steps of the knowledge updating process is skipped.

The event $E1$ is then interpreted by the victim manager VM, who evaluates the changing situation of the victim and decides that because of the serious injury, the original plan to rescue the victim has to be replaced with a new plan. A series of events ($E2$, $E3$, $E4$, $E5$, and $E6$) are then generated by the VM to indicate the change of plan. The system updates the knowledge by removing and adding corresponding actions from the PlanGraph in the association step. During the elaboration step, the system retrieves recipes for the newly added actions and apply default knowledge about intentions and capabilities from the knowledge base.

Figure 8.15 shows the PlanGraph after the planning events generated by the VM have been processed.

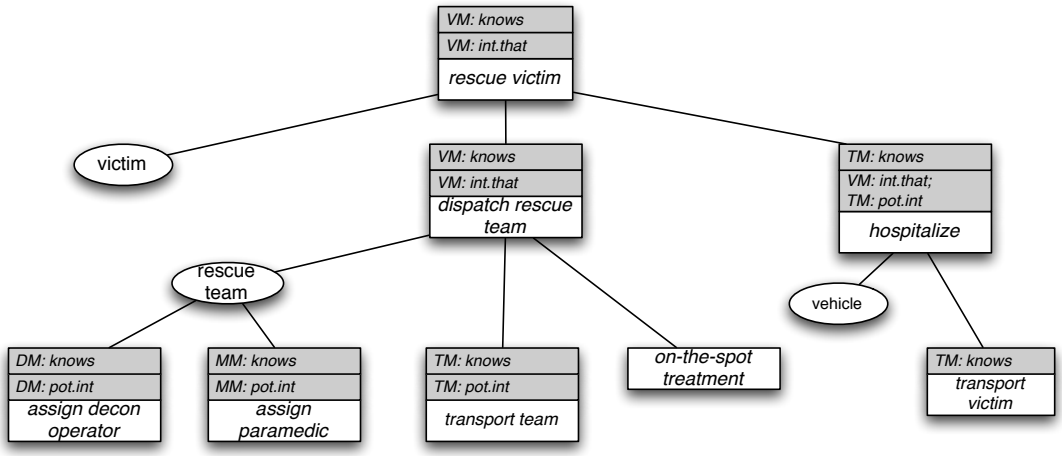


Figure 8.15. PlanGraph after $E6$ in Episode 4

After these events are distributed to the corresponding actors ($E2$ and $E5$ for the DM, $E3$, $E5$ for the MM, and $E4$, $E5$ and $E6$ for the TM), these actors interpret them, perform actions in the new plan, and generate new events ($E7$ from the DM, $E8$ from the MM, and $E9$ from the TM) that are then used to update the system's knowledge. The knowledge updating processes triggered by these events are similar to some cases in previous episodes, so we skip the details and shows the final PlanGraph after they are processed in Figure 8.16.

8.3.2 Analyzing the field of work

As we persist the PlanGraphs in the four episodes in the database during the simulation, we can use them to construct the local scopes and dependency networks following the algorithms described in Section 5.2.3 and 5.2.4. These algorithms are implemented in the EDAP server as global functions as they are frequently used by the *Visualization Service* and *Notification Module*. We use these constructed local scopes and dependency networks to analyze the three essential characteristics of the field of work as we described in our conceptual framework 2.4.2.

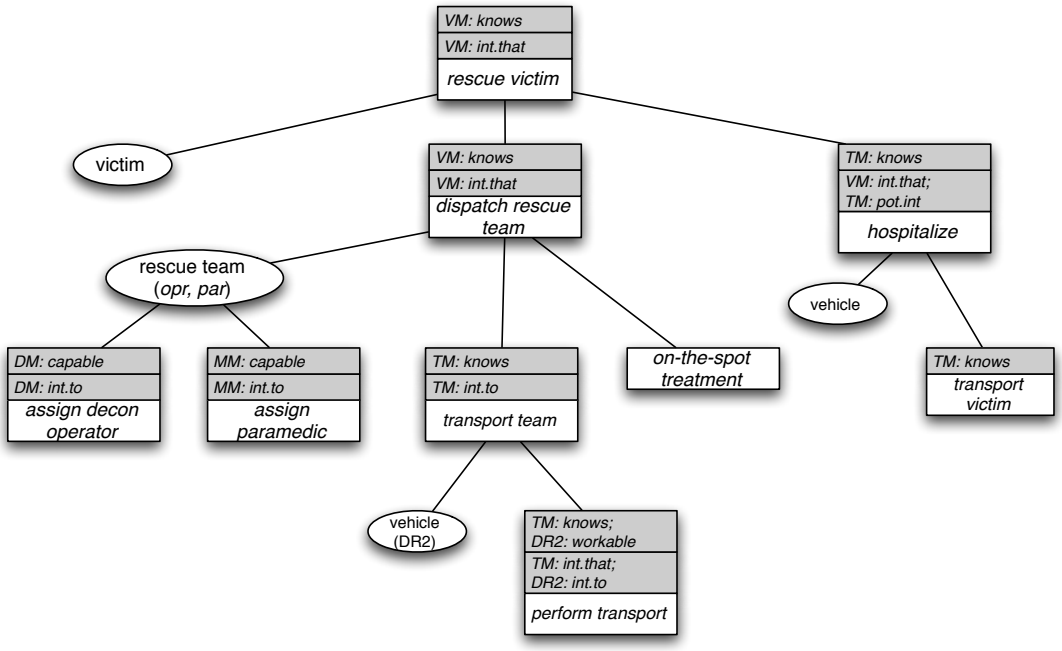


Figure 8.16. PlanGraph after Episode 4

8.3.2.1 Partiality of local scopes

One of the central concepts in our approach is the local scopes of different actors in a collaborative activity. Comparing with the filed of work representing the overall collaborative activity, the local scope of an actor represents the part of the field of work that the actor is participated in. The partiality of local scopes provides the basic for handling complexity in our approach. As the whole field of work can grow significantly in a collaborative activity, the local scopes are relatively small, and hence much more manageable for each actor.

To compare the magnitudes of local scopes and the field of work, we count the total number of entities (i.e. actions, parameters, and conditions) in the PlanGraph and the number of entities in the constructed local scope of the victim manager VM, the decontamination manager DM, the transportation manager TM, and the medical manager MM. We choose the four actors because they are involved in all the four episodes. The chart in Figure 8.17 shows the counting results. As we can see from the chart, the total number of entities in the PlanGraph (with an average of 20) is much greater than the number of entities in any of the actors'

local scopes (the average is 5.5 for VM, 6.25 for DM, 3.5 for MM, and 6.75 for TM).

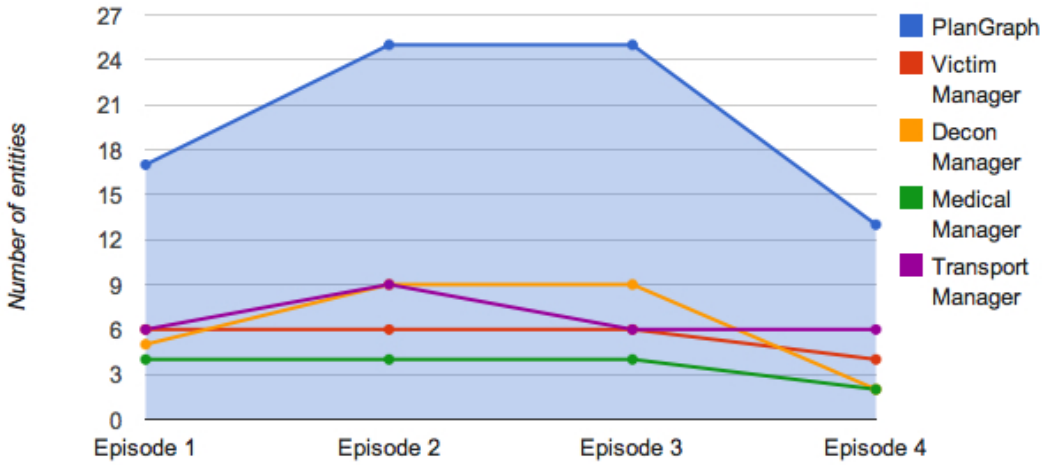


Figure 8.17. Numbers of entities in local scopes

8.3.2.2 Connectivity of local scopes

The second characteristics of the field of work is that the entities from different actors' local scopes are connected with each other, which is the basic assumption behind our event propagation mechanism. As we discussed in Section 2.4.2, there are two ways that the entities in different local scopes can be connected: through the boundary objects in overlapping local scopes and through the dependency relations.

To examine the connectivity of local scopes, we construct the dependency network from the PlanGraph after Episode 2, and map the different entities in the dependency network into the corresponding actor's local scopes. Figure 8.18 shows the visualization of the mapping result. The arrows in the figure indicate the dependency relations between entities, and the shapes in dotted lines with different colors show the local scopes of different actors. The similar mapping results can be generated for the other three episodes, however here we only focus on the Episode 2 as the PlanGraph after Episode 2 is the one with the highest level of complexity.

From this figure, we can clearly see how the entities from different local scopes can be connected through both boundary objects and dependency relations.

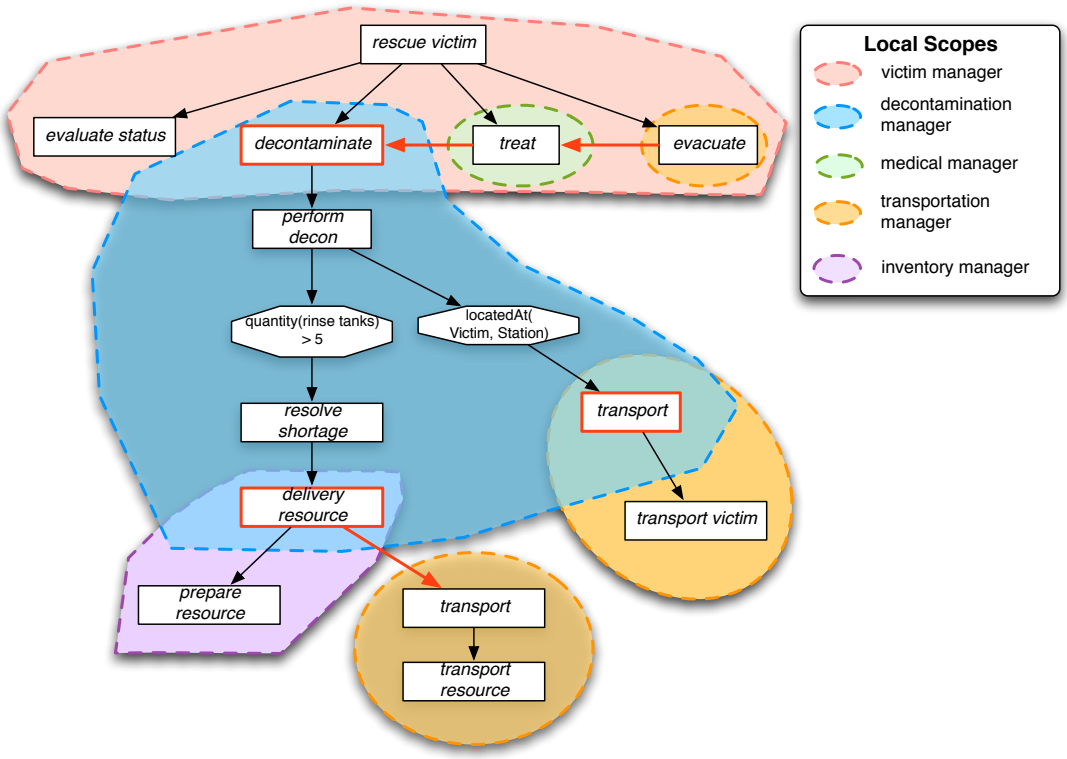


Figure 8.18. Dependency network in Episode 2 with local scopes

1. The entities with the red stroke are the boundary objects that are shared by two overlapping local scopes. For example, the *'delivery resource'* action is a shared object between the decontamination manager DM and the inventory manager IM. On one hand, the DM has the intention that the *'delivery resource'* action is performed as it is a subsidiary action for the *'resolve shortage'* action. On the other hand, the IM intends to perform the *'delivery resource'* action because the action is within her responsibility.

2. The arrows in red highlight the dependency relations that cross the boundary of two local scopes, i.e. the entity as the depender and the entity as the dependee belong to different actors' local scopes. The temporal dependency relation between the *'decontaminate'* and the *'treat'* action is such an example, as these two actions belong to the decontamination manager and the medical manager respectively. Although the two actors' local scopes do not overlap with each other, the execution state of the *'decontaminate'* action

can still have impact on the medical manager, because the *'treat'* action can only be performed after the *'decontaminate'* action is done.

8.3.2.3 Dynamics of the field of work

From the discussion in Section 8.3.1, we already see how the PlanGraph representing the field of work has been modified gradually by the events. By comparing the local scopes and dependency networks for each step, we can see that the local scopes and dependency relations are also changing in the process.

Figure 8.19 shows the changing numbers of entities in the local scope for each actor. From the chart, we can see the different actors' local scopes have quite different growing curves as the collaborative activity is developed.

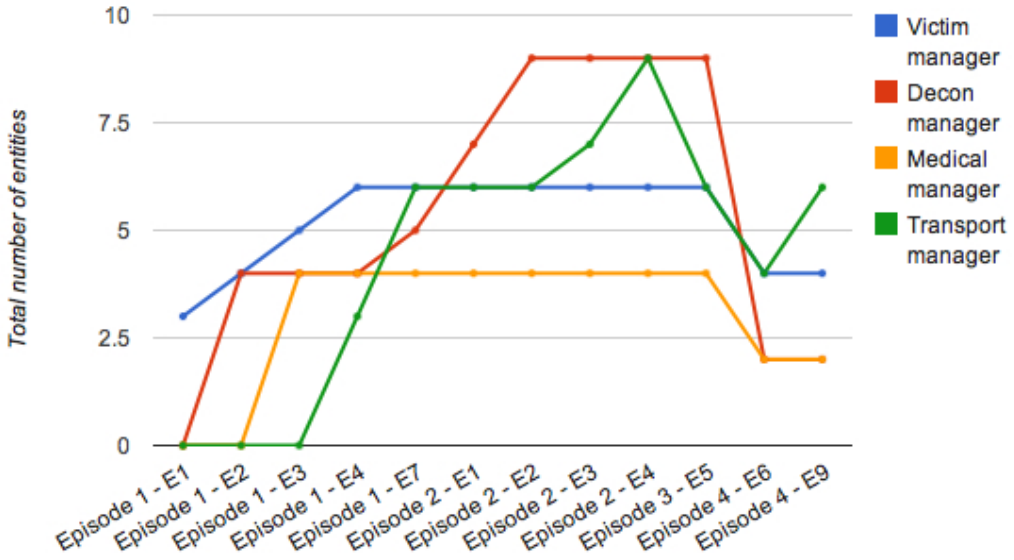


Figure 8.19. Changing numbers of entities in local scopes

Figure 8.20 shows the changing size of the dependency networks through the four episodes. For each dependency network, we measure the *size* of the network, i.e. total number of edges in a graph; and the *order* of the network, i.e. total number of vertices in a graph. The chart in the figure shows how the size of the dependency networks is changed from Episode 1 to Episode 4.

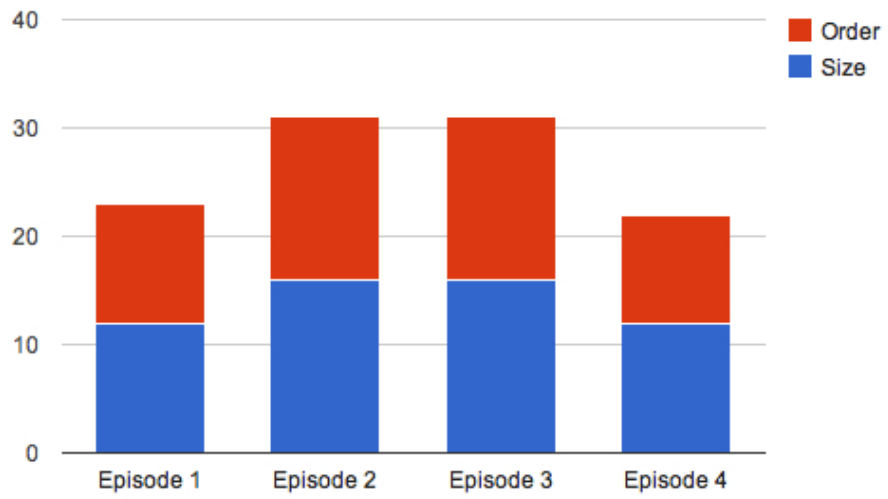


Figure 8.20. The size of dependency networks in the scenario

8.4 Promoting awareness in the scenario

The four episodes described in Section 8.2.2 demonstrate the best practices in which the events drive the development of the field of work smoothly in the scenario. However, these best practices are achievable only if the actors in the scenario could successfully develop their awareness in the process. More specifically, these episodes can only happen if the following conditions were satisfied:

1. First, the events must be distributed to the right actors at the right time . For example, Episode 1 can only happen if the VM first receives the event $E1$ indicating the discovery of a new victim, and then $E2$, $E3$, and $E4$ are consumed by the DM, MM, and TM respectively (*Event notification*).

2. Second, the events must be successfully interpreted by the receivers and hence they are able to make right decisions based on the interpretation (*Event interpretation*).

3. Last, the actors must be able to propagate the effect of an event by generating follow-up events that are later consumed by other actors, e.g. Episode 1 works only if VM can propagate the effect of $E1$ by deriving $E2$, $E3$, and $E4$ from it *Event propagation*.

To satisfy these conditions, awareness support mechanisms become very important. This section analyzes how our awareness promotion approach can be applied in the scenario to enable the satisfaction of these conditions, and compare it with existing awareness support mechanisms.

8.4.1 Supporting event notification

To enable that the events are distributed to the right actors at the right time, event notification mechanisms can be applied. As we described in Section 3.2 , existing notification mechanisms rely on event subscriptions that are managed by the human actors. The human actors register the interests in receiving certain kinds of events as subscriptions. The system filters incoming events based on the subscriptions and delivers those matched events to the actors. The event distribution in Episode 1, for example, can be achieved by a set of subscriptions that are managed by different actors. In order to receive $E1$, VM needs to subscribe to all the *'new victim'* events. In order to receive $E2$, DM needs to subscribe to the *'goal activation'* events with the condition that the goal is to decontaminate a victim. Similarly, MM needs to subscribe to the *'goal activation'* events with the condition that the goal is to treat an injured victim to receive $E3$, and TM needs to subscribe to the *'goal activation'* events with the condition that the goal is to evacuate a victim. Table 8.4 shows the set of subscriptions that need to be managed by the actors to support the event distribution in Episode 1.

Event	**Reciever**	**Subscription**
E1	Victim manager (VM)	VM needs to subscribe to all the *'new victim'* events
E2	Decontamination manager (DM)	DM needs to subscribe to the *'goal activation'* events with the condition that the goal is to decontaminate a victim
E3	Medical manager (MM)	MM needs to subscribe to the *'goal activation'* events with the condition that the goal is to treat an injured victim
E4	Transportation manager (TM)	TM needs to subscribe to the *'goal activation'* events with the condition that the goal is to evacuate a victim

E6	Transportation manager (TM)	TM needs to subscribe to the *'goal activation'* events with the condition that the goal is to transport a victim to a station
E7	Driver (DR1)	DR1 needs to subscribe to the *'resource assignment'* events with her vehicle as the assigned resource.

Table 8.4: Subscription-based event distribution in Episode 1

Alternatively, the local scope-based event notification mechanism as we propose in Section 6.1 provides an approach to distributing the events without explicit event subscriptions by the actors. In our local scope-based approach, the events are distributed to an actor if they are within the actor's local scope. For instance, the event distribution in Episode 1 can be achieved by our approach in the following way:

1. Upon receiving $E1$, the system triggers the knowledge updating process in which the system activates the goal to rescue the victim, elaborates the plan to achieve it, associates $E1$ to the *'victim'* parameter, and applies the VM's potential intention to the *'resuce'* action. The detailed explanation of this knowledge updating process can be found at Section 8.3.1.1. Then during the notification process, the system finds that $E1$ is associated with the *'victim'* parameter that falls into VM's local scope. As a result, the system believes that $E1$ is relevant and should be notified to the VM.

2. Upon receiving $E2$, the system associates the event with the *'decontaminate'* action, and applies the DM's potential intention to it. During the notification process, the system finds that $E2$ is associated with the *'decontaminate'* action that is within the DM's local scope, and therefore should be distributed to DM.

3. The similar analysis can be performed on the rest of the events in Episode 1, and the system achieves the same effect of distributing these events to the corresponding actors as the subscription-based approach.

Figure 8.21 shows how the events in Episode 1 are associated with the entities in the PlanGraph. The different actors' local scopes are also shown in the figure so that we can clearly see how these events fall into the corresponding receivers' local scopes.

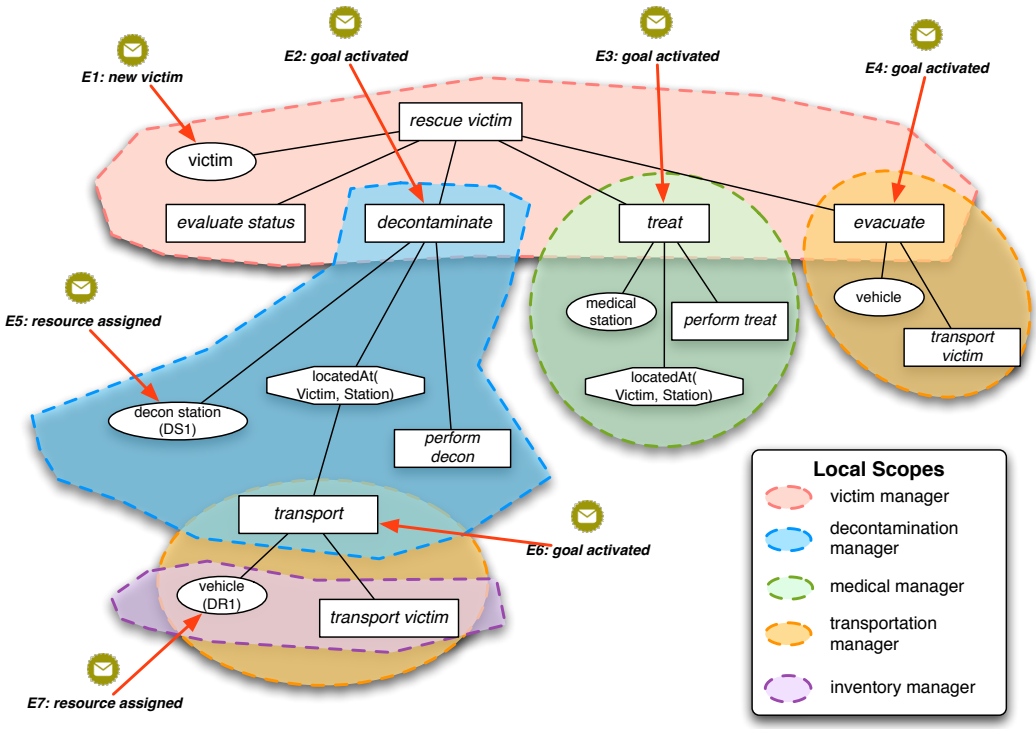


Figure 8.21. Local scope-based event notification in Episode 1

Comparing with the subscription-based approach, our local scope-based event notification mechanism has two advantages in supporting event distribution: (1) the leverage of system's reasoning to limit the human actor's effort to manage subscriptions within local scopes, and (2) the capability to handle the dynamics of the field of work. We use two concrete cases in the scenario to justify these claims.

Case 1: Reducing the effort to manage subscriptions In the first case, we consider the awareness need of the decontamination manager DM in Episode 2. Figure 8.22 shows a portion of the PlanGraph within the DM's local scope and the actions that DM's actions depend on. For example, the *'delivery resource'* action that is within the DM's local scope depends on the inventory manager IM's action

to *'prepare the resource'*. The action to transport the victim to the decontamination station depends on the driver $DR1$ to actually perform the transportation operation.

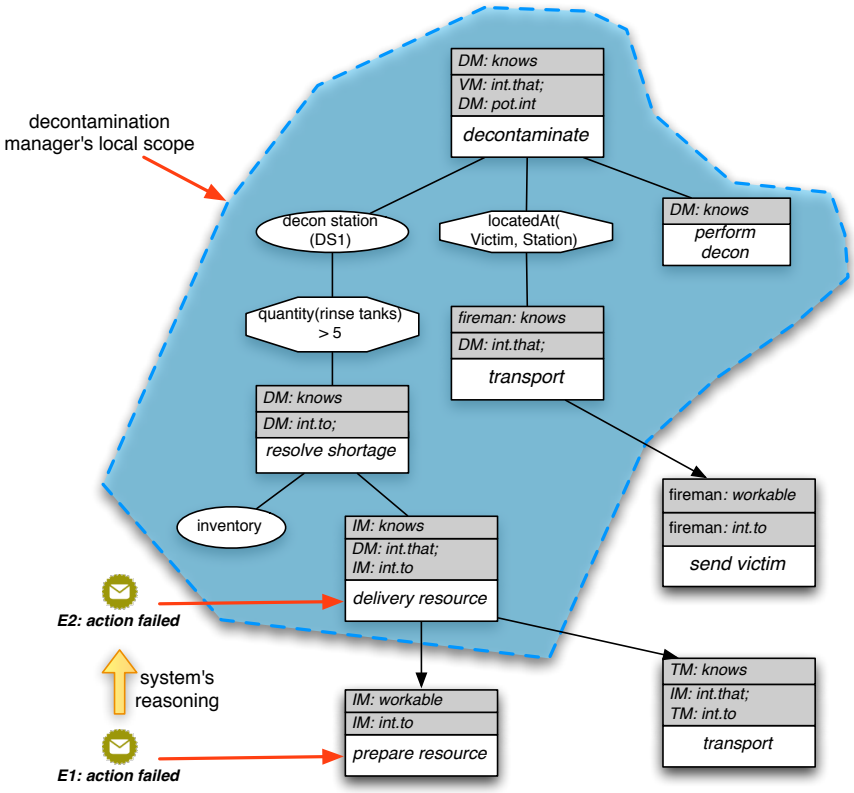


Figure 8.22. The DM's local scope in Episode 2

Now we consider an event that occurs on an action that is outside the DM's local scope. This event ($E1$) describes that the IM's action to *'prepare the resource'* failed. Although the IM's action to *'prepare the resource'* is outside the DM's local scope, such an event is relevant to the DM because the failing of this action will cause the *'delivery resource'* action that is within the DM's local scope to fail as well. Receiving this event allows the DM to evaluate the impact on her own action, for example, the DM can assign another inventory to supply the resource, or even assign the victim to another decontamination station to avoid the problem.

In the subscription-based approaches, this means that the DM has to subscribe to the events that may occur on the entities outside her local scope, i.e. the DM needs to manage the subscriptions on the IM's action to *'prepare the resource'*, the

TM's action to *'transport'* the resource, and the fireman's action to *'send victim'* to the station.

In our approach, on the other hand, the system performs the reasoning on $E1$ during the knowledge updating process, which allows the system to generate an anticipatory event $E2$ to indicate the potential failing on the *'delivery resource'* action, and attach $E2$ to the event chain along with $E1$. During the local scope-based notification process, because $E2$ falls into DM's local scope, the event chain is then distributed to the DM. In this way, the DM only needs to manage the subscriptions within the local scope, e.g. to specify the notification style of the state change on the *'delivery resource'* action, but leaves the task to distribute events outside the local scope to the system.

Case 2: handling the dynamics of the field of work In the second case, we consider the changing awareness need of the decontamination manager DM in the scenario. As we have shown earlier in Section 8.3.2.3, the actors' local scopes change a lot in the development of the collaborative activity. Figure 8.23 shows the changing local scope of DM throughout the four episodes. From Episode 1 to Episode 2, more entities are added to the DM's local scope, and from Episode 3 to Episode 4, the DM's local scope is significantly simplified due to the re-planning on the *'rescue'* action.

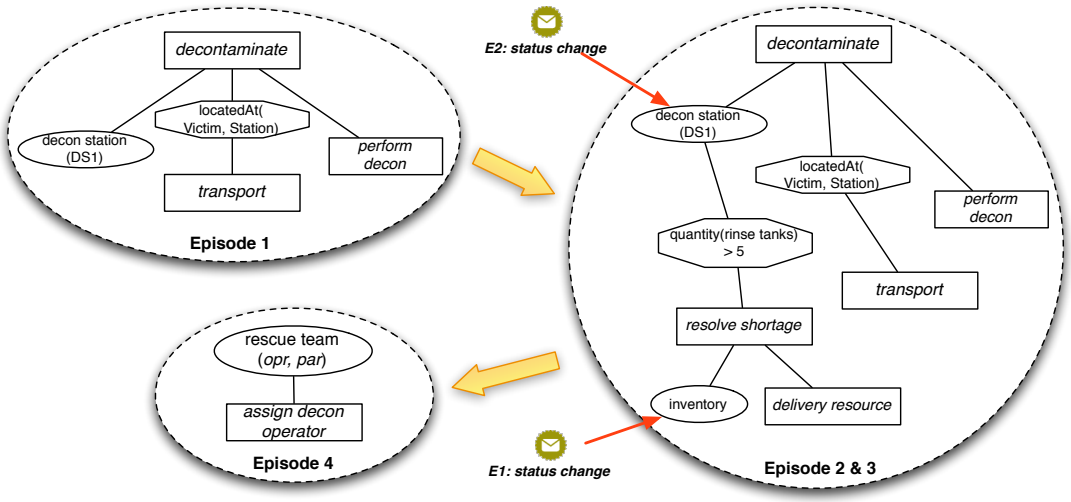


Figure 8.23. The changing local scope of the DM

Within the context of the changing local scope, we can easily see how the DM's awareness need is also changed. We consider the event $E1$ that indicates the status change of the inventory that is used to supply the rinse tanks for the station $DS1$. Such an event is only relevant to the DM in the Episode 2 and 3 when it is used to resolve the resource shortage. Similarly, the event $E2$ that indicates the status change of the decontamination station is useful information for the DM in the first three episodes, but it becomes irrelevant in Episode 4 as the DM now only focuses on assigning the operator to the rescue team.

The subscription-based notification mechanisms have difficulties to handle this kind of dynamics. In many existing systems, the subscriptions are pre-defined by the users before the collaborative activity starts. As a result, the subscriptions may be either too narrow to cover all the possible relevant events in the process, or so broad that the system notifies the user the information that has not become relevant yet. To remedy the problem, some systems allow the user to modify the subscriptions on the fly, which provides more flexibility for the users to express their changing needs. However, it requires the users to stop their current work and spend extra effort to explicitly manage the changes in their subscriptions.

Instead, Our local scope-based notification mechanism provides a better solution to handle the dynamics. As the system keeps tracking of the changing field of work, the events are always filtered out based on the current local scopes. For example, if $E1$ is generated during Episode 1, because the system is unable to associate it with any entities in DM's local scope, it is judged as irrelevant to the DM. However, if the same event occurs in Episode 2, the system will associate it with the *'inventory'* parameter that is within DM's local scope, and hence it is relevant to DM now.

8.4.2 Supporting event interpretation

The event interpretation process plays an important role in the development of collaborative activity throughout the four episodes. On one hand, it allows the actor to consume the perceived events by understanding their meanings within the actors local scope. Meanwhile, by predicting the future states of entities in the field of work, the actor can generate new events that drive the development of the

collaborative activity.

In the scenario, two types of reasoning tasks can be identified in the event interpretation process:

1. **Backward tracking to understand the origin of an event.** During the event interpretation process, the actors frequently review the historical development of an event to understand where the event comes from. The backward tracking allows the actors to dig into the reasons behind the perceived events to support their decision making. A good example of backward tracking is the fireman's interpretation of $E4$ in Episode 3. In this case, the fireman receives the execution state change event from the victim manager showing that the action to rescue the victim is delayed. Upon receiving this event, the fireman starts the backward tracking to understand that the reason for the occurrence of this event is because the action to decontaminate the victim is delayed, which in turn is because the action to transport the victim to the decontamination station is delayed. By understanding the reason behind the event, the fireman decides to send the victim directly to the decontamination station. Without the backward tracking, it is unlikely that the fireman can change the plan to perform the transportation action that is outside of her responsibility.

2. **Forward tracking to evaluate the potential impact of an event.** The forward tracking of an event allows the actors to predict the future states of their actions, which motivates the actors to make changes in the field of work. For instance, the decontamination manager's interpretation of $E1$ in Episode 2 starts with the forward tracking to evaluate the potential impact of the reduced quantity of rinse tanks in the station. During the forward tracking, the decontamination manager infers that the station cannot work properly due to this resource shortage, which in turn will impact the action to decontaminate the victim in the future. The forward tracking of the event motivates the actor to fix the problem caused by the event and activates the new action to resolve the resource shortage.

In our approach, both the backward and forward tracking can be supported

through the interlinked event view and activity view as we proposed in Section 6.2.

Supporting backward tracking The capability of the system to support backward tracking is based on the event propagation tree to keep track of the event development. When an actor generates a new event upon interpreting an existing event, a reference to the current event under interpretation will automatically attached to this new event so that the system can insert it to the event propagation tree. In this way, the system allows the actors to navigate through the event view backward to understand the origin. Figure 8.24 shows the sequence of the events that the fireman needs to look into during the interpretation of $E4$ in Episode 3, and how they are linked to the fireman's activity view.

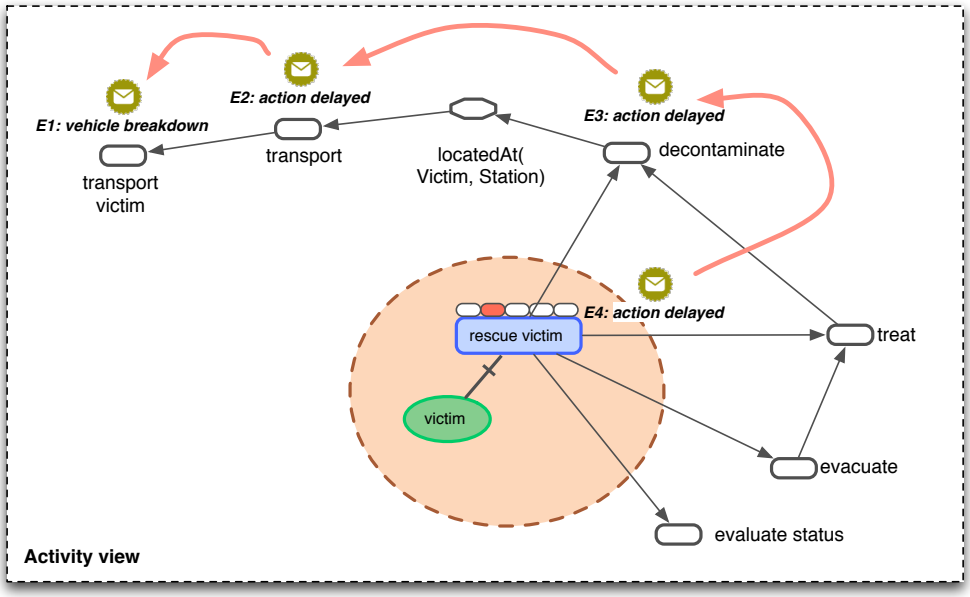


Figure 8.24. Backward tracking in the interpretation of $E4$ in Episode 3

Supporting forward tracking The forward tracking is supported by the system's capability to perform the reasoning tasks for the human actors. During the knowledge updating process, the system evaluates the consequences of the event and predicts the future states on other activities in the propagation step, and generates the event chain. As we described in Section 8.3.1.2, upon receiving $E1$ in

Episode 2, the system evaluates the conditions associated with the parameter, and finds that the condition that the quantity of the rinse tanks should be above 5 can no longer be satisfied because of this event. As a result, a new derived event is generated by the system to indicate the state change of this condition (from *'holding'* to *'open'*). Then the system performs the Bayesian network-based reasoning in the propagation step and finds that the state change of this condition (from *'holding'* to *'open'*) will lead to the failing of the *'decon station'* parameter, which may be further propagated to the upper-level *'decontaminate'* action. In this way, the event chain is formed that allows the actors to navigate through the event view forward to predict future states. Figure 8.25 shows the sequence of the events that the decontamination manager needs to look into during the interpretation of $E1$ in Episode 2.

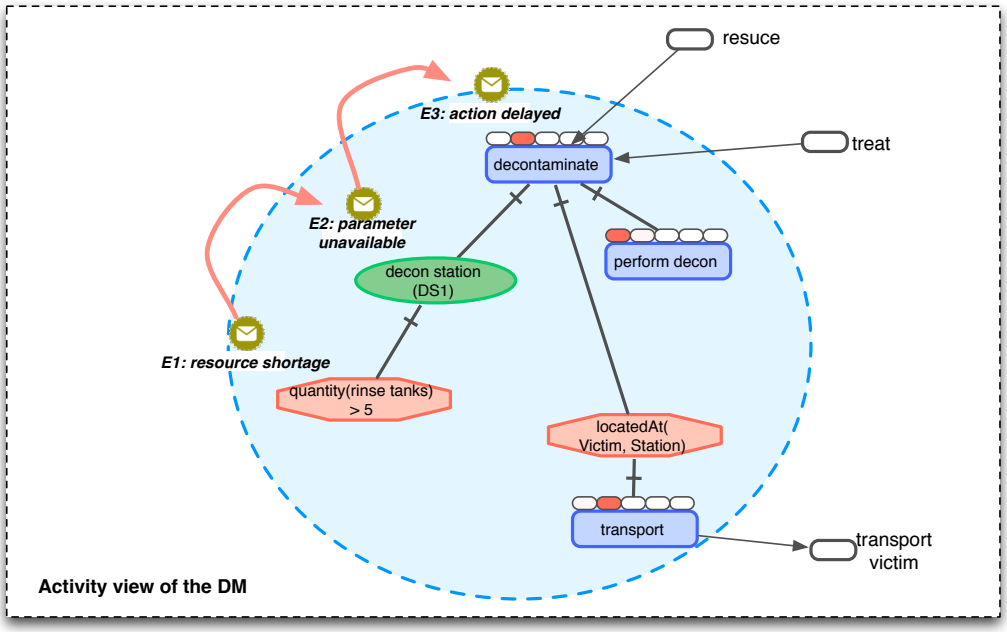


Figure 8.25. Forward tracking in the interpretation of $E1$ in Episode 2

8.4.3 Supporting event propagation

The social process of event propagation, i.e. the development of awareness involving a series of interactions among multiple actors, can be observed in all the four

episodes. The actor who receives the initial event generates his/her own interpretation of the event, which may lead to new goal activated (Episode 1&2), execution state of existing actions changed (Episode 3), or new plan generated (Episode 4). T actor externalizes the interpretation as new events, which are then received by other actors. These actors build their interpretation on top of the first actors, and may generate new events that are received by other actors. In this way, as the initial event is propagated to multiple actors, the team awareness is developed.

As we discussed in Section 6.3, our approach to support event propagation can be analyzed from both the human and the computer's perspectives. From the human perspective, it includes the functionalities to assist human actors to interpret the events built on top of each others and externalize their own interpretations as new events. From the system's perspective, it focuses on disseminating these events to the actors who are interested in further developing them.

Supporting the human effort in the event propagation process is achieved by the following functionalities provided by our approach:

1. First, the system keeps track of the event propagation process and stores the social development history of each event in the corresponding event propagation tree. The visualization of the event propagation tree and active event propagation chain in the event view allows the actors to understand who else have contributed to its development, and whose work can be impacted by the event. Figure 8.26 shows the active event propagation chains that are generated by the system at the end of each episode.

2. The capability of the human actors to externalize the results of their interpretation as new events is supported by directly manipulating the nodes in the activity view. The EDAP client described in Section 7.3 demonstrates the functionality to support the externalization. In addition, the interface for the actors to to control the visibility of their newly generated events is also supported in the client.

On the other hand, the capability of system to disseminate these derived events to relevant actors is enabled by the connectivity of local scopes and the local scope-based event notification mechanism. The analysis in 8.3.2.2 clearly shows

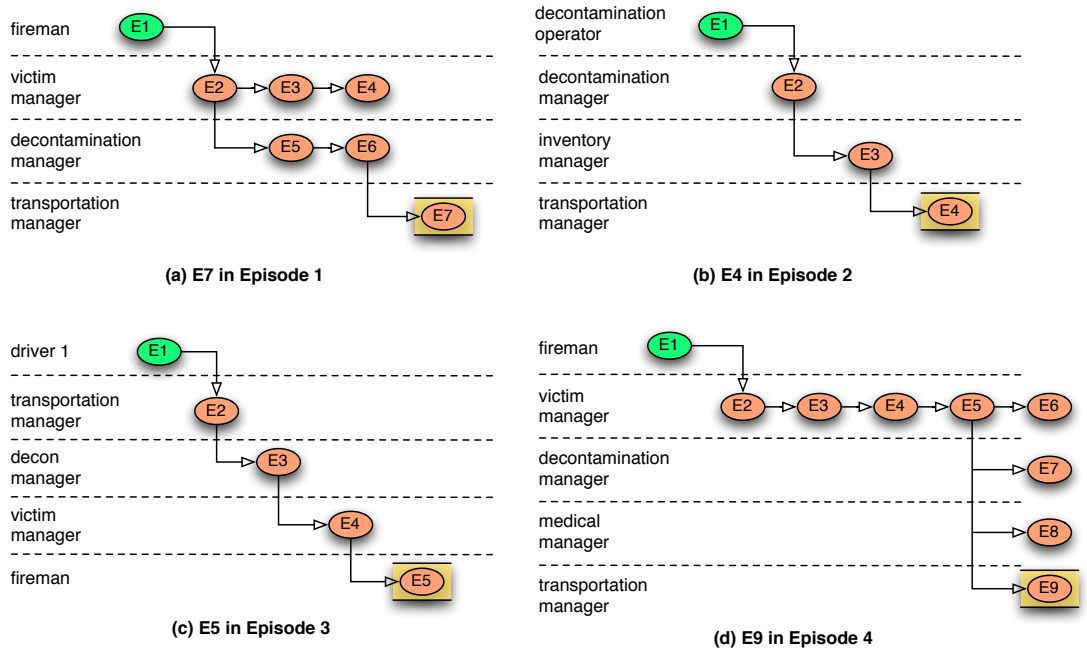


Figure 8.26. Active event propagation chains in the episodes

that the entities from different actors' local scopes can be connected with each other, either through the boundary objects in overlapping local scopes or through the dependency relations. The connectivity of local scopes allows the system to propagate the events using the local scope-based event notification mechanism. As long as the new events generated by an actor are associated with the entities that are shared with another actor, or are the dependee of another actor's actions, these events will be propagated to the other actor.

In the scenario, event propagation through both shard boundary objects and the dependency relations are evident. Figure 8.27 illustrates the two cases of event propagation in the scenario.

The first case happens in Episode 3 when $E2$ is generated by the transportation manager TM to report the delay of the action to *'transport'* the victim. Such an event is the result of the TM's interpretation of the mechanical breakdown of the $DR1$'s vehicle, and is later propagated to the decontamination manager DM. In this case, the event propagation happens because the action to *'transport'* the victim is a shared boundary object between both the DM and TM's local scopes. As $E2$ is associated with this boundary object, it falls into the DM's local scope,

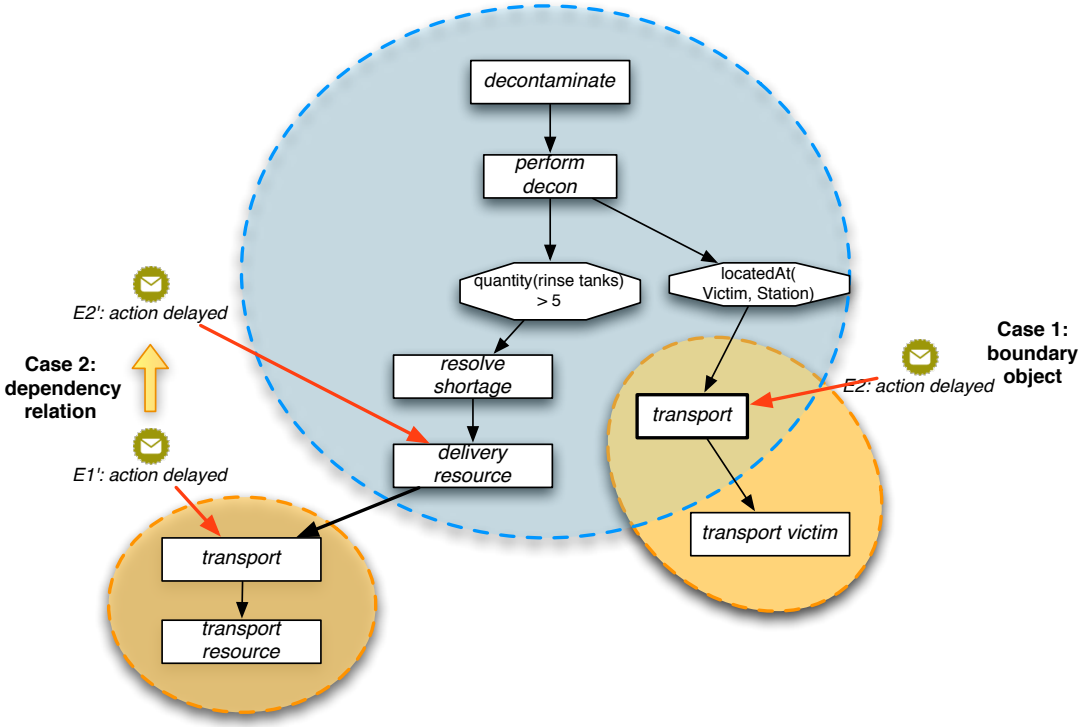


Figure 8.27. Event propagation through boundary objects and dependency relations

and as a result, our notification mechanism distributes the event to DM.

In the second case, we consider a similar case, but this time the event $E1'$ is generated by the transportation manager TM to report the delay of the action to *'transport'* the resource. In this case, the TM's action to *'transport'* the resource is outside the DM's local scope, however such an event is still relevant to the DM because the dependency relation between the *'transport'* action and the *'prepare the resource'* action. The system performs the reasoning on $E1'$ during the knowledge updating process, which allows the system to generate an anticipatory event $E2'$ to indicate the potential delay on the *'delivery resource'* action, and attach $E2'$ to the event chain along with $E1'$. During the local scope-based notification process, because $E2'$ falls into DM's local scope, it will still be notified to the DM. In this case, $E1'$ is propagated to DM because of the dependency relation connecting the two actors' local scopes.

8.5 Summary and Discussion

This chapter describes a case study in the domain of emergency response to justify our awareness promotion approach.

In Section **??**, we first describe the characteristics of the emergency response operations and the current status of awareness support in the domain. With the high level complexity and contingency, emergency response fits well into the scope of collaborative activities discussed in this study. The domain of emergency response has foundations in event driven task accomplishment that makes the event-driven awareness support more appropriate. The limitations of existing event-based models as we discussed in previous chapters also exist in the emergency response domain, which motivates our approach.

The four episodes described in Section **??** demonstrate the different types of contingency that may occur in the emergency response scenario, and how the complexity of the collaborative activity is growing with the contingency through the event-driven development of the field of work. On one hand is the events triggering the development of the field of work, and on the other hand is the changes to the field of work generate new events. It is within this recursive cycle between the events and the field of work that more actors get involved in the collaboration and their plans of actions are developed.

To evaluate the expressiveness of the PlanGraph model to represent the field of work, and the validity of the knowledge updating processes, we use the EDAP platform to simulate the event-driven development in the four episodes (Section **??**). The development of the PlanGraph model in the four episodes shows the capability of the system to keep track of the event-driven development of the field of work. Our system is able to handle the four typical development trajectories of collaborative activities as represented by the four episodes respectively.

1. Episode 1 shows the capability of the system to handle the top-down action decomposition as triggered by goal activation.

2. Episode 2 shows how the system can update the PlanGraph according to the actor's opportunistic plan development driven by unexpected events.

3. Episode 3 demonstrates the system's capability to project the impact of an

event throughout the PlanGraph, and keep track of the human actors' social development of an event, i.e. tracking the event propagation process.

4. Episode 4 shows how the system can handle the opportunistic re-planning and modify the PlanGraph to accommodate the changes.

Along with the development of the PlanGraph in the episodes, we construct the corresponding local scopes and dependency networks to analyze the expressiveness of the PlanGraph model to represent the field of work in the scenario. The results of our analysis provide the evidence that the three characteristics of the field of work as we conceptualized in Section 2.4 can be supported by the PlanGraph model.

1. The difference between the total number of entities in the PlanGraph and the numbers of entities in different actors' local scopes shows the partiality of the local scopes. Comparing with the entities in the whole field of work represented by the PlanGraph, each local scope is relatively small, and only includes a small set of the entities.

2. The mapping of local scopes into the dependency network clearly shows the entities from different actors' local scopes are connected with each other, either through the boundary objects in overlapping local scopes or through the dependency relations.

3. The numbers of entities in the local scopes and numbers of dependency relations keep changing in the development of the four episodes, which makes the dynamics of the field of work evident.

In Section **??**, we apply the awareness promotion approach in the context of the scenario to analyze the utility of our approach to support the awareness processes in the four episodes.

1. The local scope-based event notification mechanism shows two advantages in supporting event presentation in the scenario: (1) the leverage of system knowledge to limit the human actors effort to manage subscriptions within local scopes, (2) and the capability to handle dynamics. Because of the

partiality of local scopes, our approach reduces the complexity of the event notification problem since the human actors only need to manage subscriptions within their local scopes. On the other hand, the capability of our approach to model dynamics in the field of work provides a more adaptive way to judge the relevance of the events. Instead of relying on pre-defined subscriptions, the relevance of an event is evaluated based on the relation to the up-to-date local scopes in the changing field of work.

2. Our visualization framework for event interpretation supports two types of reasoning tasks that can be identified in the scenario: (1) backward tracking to understand the origin of an event, and (2) forward tracking to evaluate the potential impact of an event. The former is supported by the capability of the system to keep track of the event development in the knowledge updating process, so that the system can provide the event view to visualize the historical development of an event. The forwarding tracking is supported by the systems capability to perform the reasoning tasks for the human actors. The system's evaluation of an event's consequences and prediction of future states provide suggestions for the human actors to interpret the event.

3. Our approach to supporting event propagation includes (1) the functionalities to assist human actors to interpret the events built on top of each others and externalize their own interpretations as new events, and (2) the dissemination of events to relevant actors. The former emphasizes the social aspect of the event propagation process, as how the events are collectively generated and interpreted by multiple actors. The latter is enabled by the connectivity of local scopes in the field of work. The shared boundary objects within the overlapping local scopes and the dependency relations between entities in different actors' local scopes provide the basis for the system to propagate events using the local scope-based event notification mechanism.

Chapter 9

Conclusion

9.1 Contributions

9.2 Comparison with Existing Studies

9.3 Future Work

Appendix A

Episode Description in the Emergency Response Scenario

A.1 Episode 1: Goal activation

Actor	Type	Description
Fireman	Generate event (E1)	E1: A new victim has been found at a given location within the impacted area
Victim manager (VM)	Consume event (E1)	Upon receiving E1, VM activate the goal that the victim needs to be rescued, and decomposes the action to rescue the victim into sub-actions.
	Perform action	In order to rescue the victim, VM first evaluates the status of the victim. Because the radiation level of the victim exceeds the threshold, the victim needs to be first decontaminated. Because the victim is also physically injured, the victim needs to be treated at a medical station after decontamination. After the treatment, the victim needs to be evacuated to a shelter for recovery.

	Generate events (E2, E3, E4)	E2: A new goal to decontaminate the victim is activated E3: A new goal to treat the victim is activated E4: A new goal to evacuate the victim is activated
Decontamination manager (DM)	Consume event (E2)	Upon receiving E2, DM infers that the action to decontaminate is within her responsibility and she is capable of performing it, so she is committed to performing the action, decompose the action into sub-actions.
	Perform action	To perform decontamination, DM first needs to assign a decontamination station to the victim. After the assignment of station (DS1) to the victim, the DM activates the goal to transport the victim to the station.
	Generate event (E5, E6)	E5: The station (DS1) has been assigned to the decontamination action on victim. E6: A new goal to transport the victim to the assigned decontamination station is activated
Medical manager (MM)	Consume event (E3)	Upon receiving E3, MM infers that the action to treat the victim is within her responsibility. However, because the action depends on the action to decontaminate the victim to be completed first, she has to wait for now.
Transportation manager (TM)	Consume event (E4)	Upon receiving E4, TM infers that the action to evacuate the victim is within her responsibility. However, because the action depends on the action to treat the victim to be completed first, she has to wait for that first.
	Consume event (E6)	Upon receiving E5, TM infers that the action to transport the victim to the station is within her responsibility and she is capable of performing it, so she is committed to performing the action, and further decompose the action.

	Perform action	TM first needs to assign a driver (DR1) with a rescue vehicle to the action. Because the actual transportation action is out of the DM's responsibility, she is waiting for the driver to perform it.
	Generate event (E7)	E7: The driver (DR1) has been assigned to the action to transport the victim
Driver (DR1)	Consume event (E7)	Upon receiving E6, DR1 infers that the action to transport the victim to the station is within her responsibility and she is capable of performing it, so she starts to perform the action.

Table A.1: Episode 1 in the emergency response scenario

A.2 Episode 2: Plan development

Actor	**Type**	**Description**
Operator at the decontamination station	Generate event (E1)	E1: The quantity of available rinse tanks in the station (DS1) is running low.
Decontamination manager (DM)	Consume event (E1)	Upon receiving E1, DM infers that the station (DS1) cannot work properly due to this resource shortage. In order to ensure the victim is successfully decontaminated, a new action to resolve the resource shortage is added to the current plan to rescue the victim.
	Perform action	DM first needs to locate an inventory where the rinse tanks can be supplied. After identification of the inventory (INV1), the DM activates the goal to deliver the resource from the inventory to the station.
	Generate event (E2)	E2: A new goal to deliver the resource from (INV1) to the station (DS1) is activated.
Inventory manager (IM)	Consume event (E2)	Upon receiving E2, IM starts a new action to deliver the resource.

	Perform action	IM first needs to prepare the resource for delivery. Once the resource is ready, she activates the goal to transport the resource to the station (DS1)
	Generate event (E3)	E3: A new goal to transport the resource to the station (DS1) is activated
Transportation manager (TM)	Consume event (E3)	Upon receiving E3, TM starts a new action to transport the resource.
	Perform action	TM first needs to assign a driver (DR2) with a rescue vehicle to the action. Because the actual transportation action is out of the DM's responsibility, she is waiting for the driver to perform it.
	Generate event (E4)	E4: The driver (DR2) has been assigned to the action to transport the victim
Driver (DR2)	Consume event (E4)	Upon receiving E4, DR2 infers that the action to transport the victim to the station is within her responsibility and she is capable of performing it, so she starts to perform the action.

Table A.2: Episode 2 in the emergency response scenario

A.3 Episode 3: Responsibility transfer

Actor	**Type**	**Description**
Driver (DR1)	Generate event (E1)	E1: The rescue vehicle with the driver (DR1) encounters a mechanical breakdown.
Transportation manager (TM)	Consume event (E1)	Upon receiving E1, TM infers that because of the vehicle breakdown, the DR1's action to transport the victim cannot be achieved.
	Perform action	To solve the problem, TM attempts to assign another driver to the action. However, because all the drivers are currently in duty, the TM cannot find a driver to perform the action until a later time.

	Generate event (E2)	E2: The action to transport the victim has to be delayed because of the unavailability of drivers.
Decontamination manager (DM)	Consume event (E2)	Upon receiving E2, DM infers that because of the delay to transport the victim, the action to decontaminate the victim is also delayed.
	Generate event (E3)	E3: The action to decontaminate the victim will be delayed because of the delay of the transportation task.
Victim manager (VM)	Consume event (E3)	Upon receiving E3, VM infers that because of the delay to transport the victim, the action to rescue the victim is delayed.
	Generate event (E4)	E4: The action to rescue the victim is delayed because of the delay of the transportation task.
Fireman	Consume event (E4)	Upon receiving E4, the fireman infers that although delivery of the victim is not in the scope of her responsibility, however she can help the rescue action by sending the victim to the decontamination station (DS1).
	Generate event (E5)	E5: The fireman activates the goal to send the victim to the decontamination station (DS1).
Transportation manager (TM)	Consume event (E5)	Upon receiving E5, TM infers that because of the new action of the fireman, the responsibility of transporting the victim is now transferred from her to the fireman.

Table A.3: Episode 3 in the emergency response scenario

A.4 Episode 4: Opportunistic re-planning

Actor	**Type**	**Description**
Fireman	Generate event (E1)	E1: The physical injury of the victim becomes severe and prevents the fireman from moving the victim into the vehicle.

Victim manager (VM)	Consume event (E1)	Upon receiving E1, VM evaluates the situation of the victim and decides that because of the serious injury, the original plan to rescue the victim is no longer appropriate. A new plan needs to be adopted: a special team needs to be dispatched to the victim's location to decontaminate and treat the victim on spot, and then the victim is directly sent to the hospital for further treatment.
	Generate events (E2, E3, E4, E5, E6)	E2: The current goal to decontaminate the victim is deactivated E3: The current goal to treat the victim is deactivated E4: The current goal to evacuate the victim is deactivated E5: A new goal to dispatch a special team to the victim is activated E6: A new goal to send the victim to the hospital is activated
Decontamination manager (DM)	Consume event (E2)	Upon receiving E2, DM removes the action to decontaminate the victim from her current responsibility.
	Consume event (E5)	Upon receiving E5, DM infers that in order to perform the action to dispatch a special team to the victim, a decontamination operator needs to be assigned to the team sent to the victim.
	Perform action	By reviewing the available operators, DM decides on the operator that will be sent to the victim
	Generate event (E7)	The decontamination operator has been assigned to the special team to the victim.
Medical manager (MM)	Consume event (E3)	Upon receiving E3, MM removes the action to treat the victim from her current responsibility.
	Consume event (E5)	Upon receiving E5, DM infers that in order to perform the action to dispatch a special team to the victim, a paramedic needs to be assigned to the team sent to the victim.

	Perform action	By reviewing the available actors, DM decides on the paramedic that will be sent to the victim
	Generate event (E8)	The paramedic has been assigned to the special team to the victim.
Transportation Manager	Consume event (E4)	Upon receiving E4, TM removes the action to evacuate the victim from her current responsibility.
	Consume event (E5)	Upon receiving E5, TM infers that in order to perform the action to dispatch a special team to the victim, the action to transport the team to the victim needs to be activated.
	Perform action	TM first needs to assign a driver with a rescue vehicle to the action. Because the deactivation of the original decontamination action, the driver (DR2) who was assigned for delivering the equipment is now available for this action.
	Generate event (E9)	E9: The driver (DR2) has been assigned to the action to transport the special team to the victim
	Consume event (E6)	Upon receiving E6, TM infers that the action to send the victim to hospital is within her responsibility. However, because the action depends on other actions to be completed first, she has to wait for those first.
Driver (DR2)	Consume event (E9)	Upon receiving E9, DR2 starts to perform the action to transport the special team to the victim.

Table A.4: Episode 4 in the emergency response scenario

Bibliography

[1] Marilyn Jager Adams, Yvette J. Tenney, and Richard W. Pew. Situation Awareness and the Cognitive Management of Complex Systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(1):85–104, March 1995.

[2] Rosa Alarcón and David Fuller. Intelligent Awareness in Support of Collaborative Virtual Work Groups. In Jörg Haake and José Pino, editors, *Groupware: Design, Implementation, and Use*, volume 2440 of *Lecture Notes in Computer Science*, pages 369–384. Springer Berlin / Heidelberg, 2002.

[3] J F Allen and G Ferguson. Actions and events in interval temporal logic. *Journal of logic and computation*, 4(5):531, 1994.

[4] Gennady Andrienko, Natalia Andrienko, and Marco Heurich. An event-based conceptual model for context-aware movement analysis. *International Journal of Geographical Information Science*, 25(9):1347–1370, September 2011.

[5] P Antunes, C Sapateiro, J Pino, V Herskovic, and S Ochoa. Awareness Checklist: Reviewing the Quality of Awareness Support in Collaborative Applications. *Collaboration and Technology*, pages 202–217, 2010.

[6] H Artman and C Garbis. Situation awareness as distributed cognition. In *Proceedings of ECCE*, volume 98, 1998.

[7] Gregory Bedny and David Meister. Theory of Activity and Situation Awareness. *International Journal of Cognitive Ergonomics*, 3(1):63–72, January 1999.

[8] Steve Benford and Lennart Fahlén. A spatial model of interaction in large virtual environments. pages 109–124, September 1993.

[9] Steve Benford, Chris Greenhalgh, Tom Rodden, and James Pycock. Collaborative virtual environments. *Communications of the ACM*, 44(7):79–85, July 2001.

[10] Brandon Bennett, Anthony G Cohn, Frank Wolter, and Michael Zakharyaschev. Multi-Dimensional Modal Logic as a Framework for Spatio-Temporal Reasoning. *Applied Intelligence*, 17(3):239–251.

[11] R Bentley and T Horstmann. Supporting collaborative information sharing with the World Wide Web: The BSCW shared workspace system. *Proceedings of the 4th Int. WWW Conference*, 1995.

[12] Thmoas Berlage and Markus Sohlenkamp. Visualizing Common Artefacts to Support Awareness in Computer-Mediated Cooperation. *Computer Supported Cooperative Work (CSCW)*, 8(3):207–238, 1999.

[13] Ann Blandford and B.L. William Wong. Situation awareness in emergency medical dispatch. *International Journal of Human-Computer Studies*, 61(4):421–452, October 2004.

[14] Susanne Bodker. Computers in Mediated Human Activity. *Mind, Culture, and Activity*, 4(3):149–158, July 1997.

[15] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.

[16] Scott M Brown, Eugene Santos, and Jr. Active User Interfaces. Technical report, Intelligent Distributed Information Systems Laboratory, University of Connecticut, 1999.

[17] F Cabitza, M P Locatelli, and C Simone. Promoting Process-Based Collaboration Awareness to Integrate Care Teams. In *Business Process Management Workshops*, pages 385–396. Springer, 2009.

[18] Guoray Cai, Hongmei Wang, and Alan MacEachren. Communicating Vague Spatial Concepts in Human-GIS Interactions: A Collaborative Dialogue Approach. volume 2825 of *Lecture Notes in Computer Science*, pages 287–300. Springer Berlin / Heidelberg, 2003.

[19] Guoray Cai, Hongmei Wang, Alan M. MacEachren, and Sven Fuhrmann. Natural Conversational Interfaces to Geospatial Databases. *Transactions in GIS*, 9(2):199–221, March 2005.

[20] J M Carroll, D C Neale, P L Isenhour, M B Rosson, and D S McCrickard. Notification and awareness: synchronizing task-oriented collaborative activity. *International Journal of Human-Computer Studies*, 58(5):605–632, 2003.

[21] John M Carroll, Mary Beth Rosson, Gregorio Convertino, and Craig H Ganoe. Awareness and teamwork in computer-supported collaborations. *Interacting with Computers*, 18(1):21–46, 2006.

[22] John M. Carroll, Mary Beth Rosson, Umer Farooq, and Lu Xiao. Beyond being aware. *Information and Organization*, 19(3):162–185, July 2009.

[23] P Carstensen, T Tuikka, and C Sø rensen. Are We Done Now? Towards Requirements for Computer Supported Cooperative Software Testing. In *17th IRIS Seminar, Sy{\\"o}te, Finland*, pages 6–9, 1994.

[24] Marcelo Cataldo, Patrick A Wagstrom, James D Herbsleb, and Kathleen M Carley. Identification of coordination requirements: implications for the Design of collaboration and awareness tools. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 353–362, New York, NY, USA, 2006. ACM.

[25] R. Chen, R. Sharman, H.R. Rao, and S.J. Upadhyaya. Coordination in emergency response management. *Communications of the ACM*, 51(5):66–73, 2008.

[26] K Crowston. A taxonomy of organizational dependencies and coordination mechanisms. *MIT Center for Coordination Science Working Paper*, 1994.

[27] Nikunj P. Dalal and George M. Kasper. The design of joint cognitive systems: the effect of cognitive coupling on performance. *International Journal of Human-Computer Studies*, 40(4):677–702, April 1994.

[28] P Dourish and V Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 107–114. ACM New York, NY, USA, 1992.

[29] Paul Dourish and Sara Bly. Portholes: supporting awareness in a distributed work group. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*, pages 541–547, New York, New York, USA, June 1992. ACM Press.

[30] M J Egenhofer. Deriving the composition of binary topological relations. *Journal of Visual Languages and Computing*, 5(2):133–149, 1994.

[31] M R Endsley. Measurement of situation awareness in dynamic systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(1):65–84, 1995.

[32] M R Endsley, W B Jones, K Schneider, M McNeese, and M Endsley. A model of inter-and intrateam situational awareness: implications for design, training, and measurement. *New Trends in Cooperative Activities Understanding System Dynamics in Complex Environments*, pages 46–67, 2001.

[33] Mica R. Endsley. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(1):32–64, March 1995.

[34] Thomas Erickson, David N. Smith, Wendy A. Kellogg, Mark Laff, John T. Richards, and Erin Bradner. Socially translucent systems: social proxies, persistent conversation, and the design of babble. In *Proceedings of the SIGCHI conference on Human factors in computing systems the CHI is the limit - CHI '99*, pages 72–79, New York, New York, USA, May 1999. ACM Press.

[35] A Espinosa, F J Lerch, R E Kraut, E Salas, and S M Fiore. Explicit vs. implicit coordination mechanisms and task dependencies: One size does not fit all. *Team cognition: Understanding the factors that drive process and performance*, pages 107–129, 2004.

[36] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Co., 2010.

[37] P T Eugster, P A Felber, R Guerraoui, and A M Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[38] Robert S. Fish, Robert E. Kraut, Robert W. Root, and Ronald E. Rice. Evaluating video as a technology for informal communication. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*, pages 37–48, New York, New York, USA, June 1992. ACM Press.

[39] Geraldine Fitzpatrick, Simon Kaplan, Tim Mansfield, David Arnold, and Bill Segall. Supporting Public Availability and Accessibility with Elvin: Experiences and Reflections. *Computer Supported Cooperative Work (CSCW)*, 11(3):447–474, 2002.

[40] A U Frank. Qualitative Spatial Reasoning about Cardinal Directions'. *Auto Carto 10: Technical Papers*, page 148, 1991.

[41] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, November 1991.

[42] Ludwin Fuchs. AREA: A Cross-Application Notification Service for Groupware. *Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work*, pages 61–80, 1999.

[43] Ludwin Fuchs, Uta Pankoke-Babatz, and Wolfgang Prinz. Supporting cooperative awareness with local event mechanisms: the groupdesk system. pages 247–262, September 1995.

[44] S R Fussell, R E Kraut, F J Lerch, W L Scherlis, M M McNally, and J J Cadiz. Coordination, overload and team performance: effects of team communication strategies. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 275–284. ACM, 1998.

[45] Tom Gross and Wolfgang Prinz. Modelling Shared Contexts in Cooperative Environments: Concept, Implementation, and Evaluation. *Computer Supported Cooperative Work (CSCW)*, 13(3):283–303, 2004.

[46] B J Grosz and S Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[47] Barbara J. Grosz and Luke Hunsberger. The dynamics of intention in collaborative activity. *Cognitive Systems Research*, 7(2-3):259–272, June 2006.

[48] Jonathan Grudin. Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, 37(1):92–105, January 1994.

[49] Carl Gutwin and Saul Greenberg. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work (CSCW)*, 11(3):411–446, 2002.

[50] Carl Gutwin, Mark Roseman, and Saul Greenberg. A usability study of awareness widgets in a shared workspace groupware system. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work - CSCW '96*, pages 258–267, New York, New York, USA, November 1996. ACM Press.

[51] Christian Heath, Marcus Sanchez Svensson, Jon Hindmarsh, Paul Luff, and Dirk vom Lehn. Configuring Awareness. *Computer Supported Cooperative Work (CSCW)*, 11(3):317–347, 2002.

[52] Mary Hegarty. The Cognitive Science of Visual-Spatial Displays: Implications for Design. *Topics in Cognitive Science*, 3(3):446–474, July 2011.

[53] I. Herman, G. Melancon, and M.S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[54] D Hernandez, E Clementini, and P Di Felice. Qualitative distances. *Spatial Information Theory A Theoretical Basis for GIS*, pages 45–57, 1995.

[55] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, March 2004.

[56] Scott E Hudson and Ian Smith. Techniques for addressing fundamental privacy and disruption tradeoffs in awareness support systems. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 248–257, New York, NY, USA, 1996. ACM.

[57] Luke Hunsberger and Charles Ortiz. Dynamic intention structures I: a theory of intention representation. *Autonomous Agents and Multi-Agent Systems*, 16(3):298–326.

[58] E Hutchins and G Lintern. *Cognition in the Wild*, volume 262082314. MIT press Cambridge, MA, 1995.

[59] G Jakobson, J Buford, and L Lewis. Towards an architecture for reasoning about complex event-based dynamic situations. In *Third International Workshop on Distributed Event-Based Systems*, page 62. Citeseer, 2004.

[60] Yasir Javed, Tony Norris, and David Johnston. Ontology-Based Inference to Enhance Team Situation Awareness in Emergency Management. In *Proceedings of the 8th International ISCRAM Conference*, number May, Lisbon, Portugal, 2011.

[61] Ken Kaneiwa, Michiaki Iwazume, and Ken Fukuda. An Upper Ontology for Event Classifications and Relations. In Mehmet Orgun and John Thornton, editors, *AI 2007: Advances in Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 394–403. Springer Berlin / Heidelberg, 2007.

[62] A Kittur, B Lee, and R E Kraut. Coordination in collective intelligence: the role of team structure and task interdependence. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1495–1504. ACM, 2009.

[63] Scaife M. and Rogers Y. External cognition: how do graphical representations work? *International Journal of Human-Computer Studies*, 45(2):185–213, 1996.

[64] P Markopoulos. A design framework for awareness systems. *Awareness Systems*, pages 49–72, 2009.

[65] D. Scott McCrickard, C. M. Chewar, Jacob P. Somervell, and Ali Ndiwalana. A model for notification systems evaluation—assessing user goals for multitasking activity. *ACM Transactions on Computer-Human Interaction*, 10(4):312–338, December 2003.

[66] D Mendonça and F Fiedrich. Design for improvisation in computer-based emergency response systems. *... on Information Systems for Crisis Response ...*, 2004.

[67] Gero Mhl, Ludger Fiege, and Peter Pietzuch. Distributed Event-Based Systems. November 2010.

[68] Susan Mohammed and Brad C. Dumville. Team mental models in a team knowledge framework: expanding theory and measurement across disciplinary boundaries. *Journal of Organizational Behavior*, 22(2):89–106, March 2001.

[69] B A Nardi. Studying context: A comparison of activity theory, situated action models, and distributed cognition. *Context and consciousness: Activity theory and human-computer interaction*, pages 69–102, 1996.

[70] U Neisser. *Cognition and reality: Principles and implications of cognitive psychology.* WH Freeman/Times Books/Henry Holt & Co, 1976.

[71] A A Nofi. Defining and measuring shared situational awareness. Technical report, DTIC Document, 2000.

[72] Donald A Norman. Design principles for cognitive artifacts. *Research in Engineering Design*, 4(1):43–50, 1992.

[73] Antti Oulasvirta, Renaud Petit, Mika Raento, and Sauli Tiitta. Interpreting and Acting on Mobile Awareness Cues. *Human–Computer Interaction*, 22(1):97–135, 2007.

[74] J Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[75] Elin Rø nby Pedersen and Tomas Sokoler. AROMA: abstract representation of presence supporting mutual awareness. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '97*, pages 51–58, New York, New York, USA, March 1997. ACM Press.

[76] P P Perla, M Markowitz, A A Nofi, C Weuve, and J Loughran. Gaming and shared situation awareness. Technical report, DTIC Document, 2000.

[77] Jens Pottebaum, Alexander Artikis, Robin Marterer, and Rainer Koch. Event Definition for the Application of Event Processing to Intelligent Resource Management. In *Proceedings of the 8th International ISCRAM Conference*, number May, Lisbon, Portugal, 2011.

[78] Wolfgang Prinz. NESSIE: An Awareness Environment for Cooperative Settings. *Proceedings of the sixth conference on European Conference on Computer Supported Cooperative Work*, pages 391–410, 1999.

[79] W O Quine. Events and reification. *Events*, pages 107–116, 1985.

[80] A B Raposo, L P Magalhães, I L M Ricarte, and H Fuks. Coordination of collaborative activities: A framework for the definition of tasks interdependencies. In *Proceedings of Seventh International Workshop on Groupware*, pages 170–179. IEEE, 2002.

[81] U Rauschenbach. Supporting awareness in shared workspaces using relevance-dependent event notifications. *Proceedings of the CVE*, 1996.

[82] Markus Rittenbruch. Atmosphere: A Framework for Contextual Awareness. *International Journal of Human-Computer Interaction*, 14(2):159–180, June 2002.

[83] Markus Rittenbruch and Gregor McEwan. An Historical Reflection of Awareness in Collaboration. *Awareness Systems*, pages 3–48, 2009.

[84] Markus Rittenbruch, Stephen Viller, and Tim Mansfield. Announcing Activity: Design and Evaluation of an Intentionally Enriched Awareness Service. *HumanComputer Interaction*, 22(1-2):137–171, 2007.

[85] Tom Rodden. Populating the application: a model of awareness for cooperative applications. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work - CSCW '96*, pages 87–96, New York, New York, USA, November 1996. ACM Press.

[86] Mark Roseman and Saul Greenberg. Building real-time groupware with GroupKit, a groupware toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1):66–106, March 1996.

[87] E Salas, C Prince, D P Baker, and L Shrestha. Situation awareness in team performance: Implications for measurement and training. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(1):123–136, 1995.

[88] P M Salmon, N A Stanton, G H Walker, D Jenkins, C Baber, and R McMaster. Representing situation awareness in collaborative systems: a case study in the energy distribution domain. *Ergonomics*, 51(3):367–84, March 2008.

[89] Paul M. Salmon, Neville A. Stanton, Guy H. Walker, Chris Baber, Daniel P. Jenkins, Richard McMaster, and Mark S. Young. What really is going on? Review of situation awareness models for individuals and teams. *Theoretical Issues in Ergonomics Science*, 9(4):297–323, July 2008.

[90] Paul M. Salmon, Neville A. Stanton, Guy H. Walker, Daniel P. Jenkins, and Laura Rafferty. Is it really better to share? Distributed situation awareness and its implications for collaborative system design. *Theoretical Issues in Ergonomics Science*, 11(1-2):58–83, January 2010.

[91] Ovidiu Sandor, Cristian Bogdan, and John Bowers. Aether: an awareness engine for CSCW. pages 221–236, September 1997.

[92] K Schmidt and L Bannon. Taking CSCW seriously. *Computer Supported Cooperative Work (CSCW)*, 1(1):7–40, 1992.

[93] Kjeld Schmidt. The Problem with 'Awareness': Introductory Remarks on 'Awareness in CSCW'. *Computer Supported Cooperative Work (CSCW)*, 11(3):285–298, 2002.

[94] Kjeld Schmidt and Carla Simonee. Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work (CSCW)*, 5(2):155–200, 1996.

[95] S Y Shen and M J Shaw. Managing coordination in emergency response systems with information technologies. *Proceedings of the Tenth Americas Conferences on Information Systems*, pages 2110–2120, 2004.

[96] Yedendra Babu Shrinivasan and Jarke J. van Wijk. Supporting the analytical reasoning process in information visualization. In *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, page 1237, New York, New York, USA, April 2008. ACM Press.

[97] Y Shu and K Furuta. An inference method of team situation awareness based on mutual awareness. *Cognition, Technology & Work*, 7(4):272–287, 2005.

[98] J S Sichman and R Conte. Multi-agent dependence by dependence graphs. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 483–490. ACM, 2002.

[99] Riyaz Sikora and Michael J Shaw. A Multi-Agent Framework for the Coordination and Integration of Information Systems. *Management Science*, 44(11):pp. S65–S78, 1998.

[100] C Simone, M Divitini, and K Schmidt. A notation for malleable and interoperable coordination mechanisms for CSCW systems. In *Proceedings of conference on Organizational computing systems*, pages 44–54. ACM, 1995.

[101] Carla Simone and Stefania Bandini. Integrating Awareness in Cooperative Applications through the Reaction-Diffusion Metaphor. *Computer Supported Cooperative Work (CSCW)*, 11(3):495–530, 2002.

[102] Kip Smith and P. A. Hancock. Situation Awareness Is Adaptive, Externally Directed Consciousness. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(1):137–148, March 1995.

[103] M Sohlenkamp, W Prinz, and L Fuchs. PoliawaC: design and evaluation of an awareness-enhanced groupware client. *AI & Society*, 14(1):31–47, 2000.

[104] MD Spiteri. *An architecture for the notification, storage and retrieval of events*. PhD thesis, University of Cambridge, 2000.

[105] N A Stanton, R Stewart, D Harris, R J Houghton, C Baber, R McMaster, P Salmon, G Hoyle, G Walker, M S Young, M Linsell, R Dymott, and D Green. Distributed situation awareness in dynamic systems: theoretical development and application of an ergonomics methodology. *Ergonomics*, 49(12-13):1288–311, January 2006.

[106] Neville A. Stanton, Paul M. Salmon, Guy H. Walker, and Daniel Jenkins. Genotype and phenotype schemata and their role in distributed situation awareness in collaborative systems. *Theoretical Issues in Ergonomics Science*, 10(1):43–68, January 2009.

[107] Kimberly Tee, Saul Greenberg, and Carl Gutwin. Artifact awareness through screen sharing for distributed groups. *International Journal of Human-Computer Studies*, 67(9):677–702, September 2009.

[108] Loren G. Terveen. Overview of human-computer collaboration. *Knowledge-Based Systems*, 8(2-3):67–81, April 1995.

[109] J.J. Thomas and K.A. Cook. A visual analytics agenda. *IEEE Computer Graphics and Applications*, 26(1):10–13, January 2006.

[110] Brian Tomaszewski and Alan M. MacEachren. Geo-historical context support for information foraging and sensemaking: Conceptual model, implementation, and assessment. In *2010 IEEE Symposium on Visual Analytics Science and Technology*, pages 139–146. IEEE, October 2010.

[111] W Treurniet, K van BuulBesseling, and J Wolbers. Collaboration awarenessa necessity in crisis response coordination. In *9th International ISCRAM Conference*, Vancouver, Canada, 2012.

[112] Sebastien Truptil, Anne-Marie Barthe, Frederick Benaben, and Roland Stuehmer. Nuclear Crisis Use-Case Management in an Event-Driven Architecture. *Lecture Notes in Business Information Processing*, 99(6):464–472, 2012.

[113] M. Turoff, M. Chumer, B.V. de Walle, and X. Yao. The design of a dynamic emergency response management information system (DERMIS). *Journal of Information Technology Theory and Application (JITTA)*, 5(4), 2004.

[114] J Uhlarik and D A Comerford. A review of situation awareness literature relevant to pilot surveillance functions. Technical report, DTIC Document, 2002.

[115] Chunhua Weng and John H. Gennari. Asynchronous collaborative writing through annotations. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work - CSCW '04*, page 578, New York, New York, USA, November 2004. ACM Press.

[116] C. D. Wickens. Situation Awareness: Review of Mica Endsley's 1995 Articles on Situation Awareness Theory and Measurement. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 50(3):397–403, June 2008.

[117] Bo Yu and Guoray Cai. Using Intentions and Plans of Mobile Activities to Guide Geospatial Web Service Composition. In *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, pages 141–148. IEEE, 2010.

[118] E S K Yu and J Mylopoulos. An actor dependency model of organizational work: with application to business process reengineering. In *Proceedings of the conference on Organizational computing systems*, pages 258–268. ACM, 1993.

[119] May Yuan. Representing Complex Geographic Phenomena in GIS. *Cartography and Geographic Information Science*, 28:83–96(14), 2001.

[120] L Zelnik-Manor and M Irani. Event-based analysis of video. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II—-123. IEEE, 2001.

[121] Qixing Zheng, Kellogg Booth, and Joanna McGrenere. Co-authoring with structured annotations. In *Proceedings of the SIGCHI conference on Human Factors in computing systems - CHI '06*, page 131, New York, New York, USA, April 2006. ACM Press.

Vita

Bo Yu

Here goes the vita. To be added.