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

DESIGNING FOR AWARENESS PROMOTION IN LARGE-SCALE DISTRIBUTED 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

May 2013

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

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

Alan M. MacEachren
Professor of Geography

Mary Beth Rosson
Professor of Information Sciences and Technology

Xiaolong (Luke) Zhang
Associate Professor of Information Sciences and Technology

*Signatures are on file in the Graduate School.

Abstract

Maintaining awareness is central to effective coordination in distributed collaborative activities. Existing awareness solutions work for relatively small-scale collaboration in traditional workspace, and they suffer from either inflexibility or lack of scalability if applied to large-scale distributed collaborative activities that are characterized by higher level of complexity and dynamics, such as emergency response or medical systems. The overall objective of this study is to address the major challenges of awareness support in these large-scale and highly distributed collaborative activities.

In order to achieve the research objective, this study follows the design science paradigm, i.e. understanding of awareness problems in large-scale collaboration and their solutions are achieved through a set of design activities to develop useful and usable awareness supporting systems. Based on the grounding work in literature, we develop an integrated conceptual model to understand the awareness phenomena in large-scale distributed collaboration. This model helps us to generate concrete design issues that need to be addressed, identify knowledge gaps in existing studies, and guide the design of the computational awareness promotion framework. Following the computational framework, we develop the prototype system, EDAP, and perform a case study in emergency response to validate the approach and demonstrate its feasibility and utility.

This study has made two major contributions towards the research objective. Firstly, it provides an integrated conceptual model on top of several interrelated constructs, i.e. activity, local scope, and dependency, to understand the distributed nature of the awareness phenomena. The model is able to account for how the awareness is distributed across multiple team members through the integration of individual cognitive processes and team processes. Secondly, it presents a computational framework for awareness promotion. Comparing with existing awareness supporting systems, the awareness promotion approach emphasizes the active role of the computer to mediate awareness processes based on a formal knowledge representation of collaborative activities. The awareness promotion approach has shown several advantages to handle the increased level of complexity and dynamics in large-scale distributed collaborative activities: (1) it utilizes the computational knowledge representation to model collaborative activities and offloads

some of the representation and reasoning efforts from the human to the computer; (2) the knowledge representation is dynamically updated to reflect current state of the collaborative work, which allows it to handle a variety of dynamics; (3) as it represents the collective knowledge about collaborative activities at the team level, it can provide support to mediate awareness transactions among multiple team members.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem scope

Awareness is one of the most active research areas in computer supported cooperative work (CSCW) [26, 93, 82]. At its essence, awareness refers to the ability of collaborators to understand each other's activities and relate them to a joint context [82]. This context is essential for collaboration as it is used to ensure that individual contributions are relevant to the group's activity as a whole, and allow groups to coordinate the process of collaborative work [26].

Awareness falls into the category of 'articulation work' that is required for collaborative work but not its primary goals [92]. People want to maintain the awareness of one another to ensure their joint effort is coordinated and integrated, but at the same time they want to do it effortlessly so that it does not interrupt their current line of work [44]. This is unproblematic in ordinary face-to-face environments where the mechanics of collaboration are natural, spontaneous, and unforced [50]. However, it becomes a significant task in distributed collaboration because of the limited interaction resources available to the actors [21]. As a result, a large number of computer-supported awareness mechanisms and tools have been proposed in the literature, aiming to support awareness in distributed collaboration with minimal attention and effort from the participants of teamwork [82, 64].

Although much progress has been made in designing awareness mechanisms to support distributed collaboration at relatively small and medium scales [5], it becomes a

much more difficult task to support awareness in complex and highly distributed activities [16]. The collaborative activities of interest in this study are these large-scale distributed collaborative settings, where awareness is unlikely to be achieved effortlessly, and hence it becomes even more important to design computational artifacts to support awareness. In the rest of this section, we first define the scope of large-scale collaborative environments in this study, and then present the major challenges for awareness support in these environments that motivate this work.

1.1.1 Large-scale collaboration

In general, collaborative activities we consider in this study are a subset of synchronous and distributed collaborative settings [29], with a higher level of complexity and dynamics:

1. *Higher level of complexity*: The collaborative activities in this study usually include a large number of team workers that engage in a variety of distributed, yet interdependent actions. On one hand, the collaborators play specialized roles, possess distinct knowledge, and perform different actions in the course of joint endeavors. On the other hand, their actions are inter-connected because of a variety of dependencies that may occur.

2. *Higher level of dynamics*: The actors work in dynamic settings that entail frequent changes in the environment and their activities. As a result, plans of the collaborative activities are under continuous development and the tasks and roles of team members cannot be precisely specified in advance.

Examples of large-scale collaboration within these boundaries abound in practical applications such as emergency response [111], medical systems [14], and military training [65]. Our interest in supporting awareness is motivated by the need to support large-scale distributed collaboration in emergency response operations [18, 17]. An example of these collaborative environments under discussion can be illustrated in the following scenario.

An Emergency Response Scenario. Due to a critical accident in a nuclear plant, a large quantity of radioactive substance is accidentally released in the atmosphere. 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 covers 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 (emergency operation center, mobile command-and-control posts, medical stations, etc.). The different actors have complementary knowledge and skills, and hence divide the work so that they can work relatively autonomously within their local environment and responsibilities. In the same time, actions of different actors are interleaved with each other due to different types of dependencies [94]. For example, the decontamination action can only be performed after the victim is transported to the station, and the medical treatment can only be performed after the victim is decontaminated.

Furthermore, exceptions to the planned responses are a common and critical factor in these activities [111]. Re-planning of actions and re-allocation of resources go on as a continuous unpredictable process. When a rescue vehicle breaks down, it creates a limitation on the use of the vehicle to transport the victim, which leads to the assignment of another vehicle to the action, or even re-planning of the whole activity to rescue the victim. In such dynamic environment, what specific information is of concern and interest to a given individual is changing rapidly.

1.1.2 Challenges in awareness support

The increased level of complexity and dynamics in these collaborative environments poses major design challenges for awareness support, which have limited support in existing awareness supporting systems.

Managing increased level of complexity With low level of complexity, awareness can be achieved by means of presenting 'shared spaces' among collaborators. This can be 'shared media spaces' that provide continually audio-video links between distributed actors [27], or 'shared virtual spaces' that provide various types of virtual spaces to support awareness, such as virtual meeting rooms [13], or more popularly, 'shared workspaces' where people can see and manipulate artifacts related to their activities [50]. Despite the different forms 'shared spaces' can take, they have the same goal to make the shared work settings visible, so that people can keep an eye on what the rest of the group is doing while doing their individual work, and develop the shared awareness of the group situation [93].

However, when the complexity of the collaborative activities scales up, maintaining a shared understanding of the whole situation becomes more difficult, and less desired by the team members. First, the magnitude of the collaborative work can grow significantly as hundreds or thousands of actors engage in myriads of interdependent activities. It becomes difficult or even impossible to share all the aspects in the field of the collaborative work with every actor. Furthermore, these complex activities are usually highly distributed as team members play specialized roles and engage in different actions. Team members experience the situation in different ways, as defined by their own personal goals, roles, tasks, skills, and so on. Team members have their own awareness, related to the goals that they are working towards. Even though they may have access to the same information, differences in goals, roles, the tasks being performed make them view it differently [90]. As a result, it becomes more important for the awareness system to understand and support team members' distinct awareness requirements, rather than merely making the shared context visible.

Handling increased level of dynamics Developing appropriate awareness mechanisms must be based on a solid understanding of the team members' awareness requirements, e.g. what kinds of awareness information is relevant to each team member, or how the information should be presented. In relatively static work domains, the set of awareness requirements can be identified in advance so that the awareness system can be designed to support them. However, in these dynamic environments characterized by frequent and continuous changes in the environment and activities, team members' awareness requirements are quite fleeting. In order to adapt to these changes, the actors engage in different actions, assign and reassign resources for them, or change their ways to perform them. As the collaborative activity develops, they are likely to modify their

awareness requirements accordingly to achieve their changing goals. As a result, it would be highly desirable for the awareness systems to be able to keep track of these dynamics over time and assess the actors' changing awareness requirements as they emerge in the evolving collaborative activities.

1.2 Research approach and contributions

By setting the scene in the previous section, the overall objective of this dissertation is to address the major challenges of awareness support as distributed collaborative activities are scaled up to more complex and dynamic situations. To achieve the research objective, this study follows the design science paradigm in information systems research [56]. Knowledge and understanding of the aforementioned awareness problems in large-scale collaboration and their solutions are achieved through a set of design activities to develop useful and usable awareness supporting systems. Figure 1.1 provides the overview of the research approach in this study.

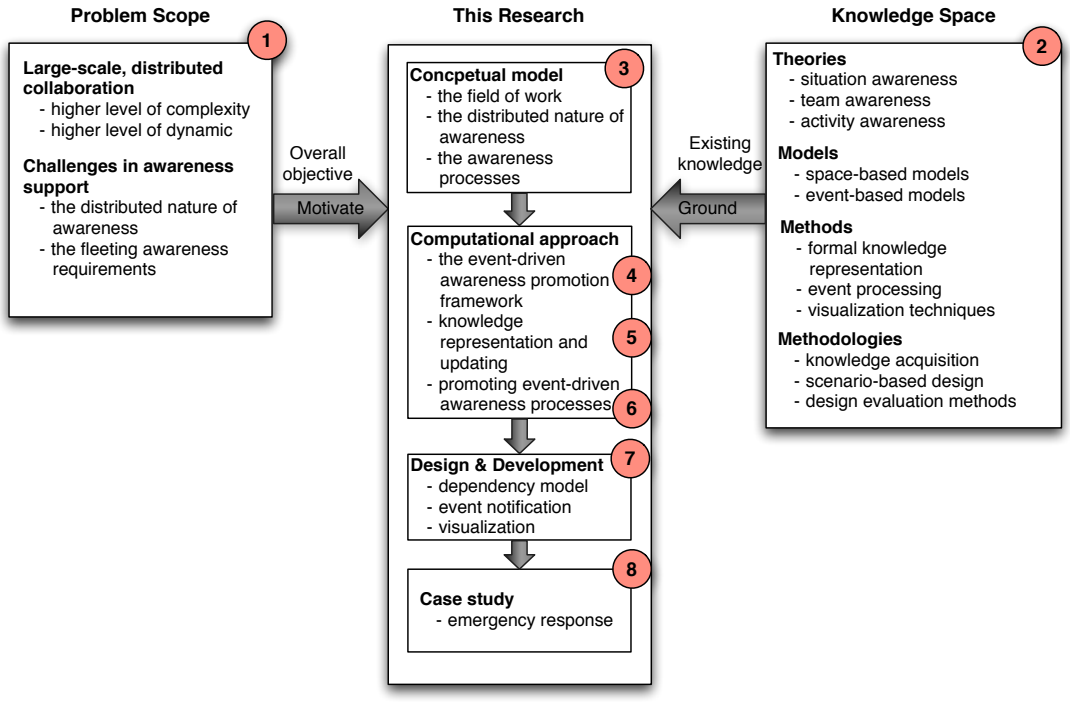


Figure 1.1. Overview of the research approach

The research starts with defining the problem scope and specifying the design challenges in the problem scope (Chapter 1). Then existing knowledge about theories and

models to understand the awareness phenomena is reviewed to provide the theoretical foundation of this study (Chapter 2). Based on the grounding work in the literature, we develop the conceptual model to understand the awareness phenomena in large-scale distributed collaboration (Chapter 3). Such a conceptual model helps us to develop concrete design issues that need to be addressed, identify knowledge gaps in existing studies, and guide our design of the computational awareness promotion framework (Chapter 4, 5, and 6). Following the computational approach, we develop the prototype system to prove the feasibility of the approach (Chapter 7). Then we perform the case study in a concrete emergency response scenario to demonstrate the utility of our approach in supporting awareness in large-scale distributed collaboration (Chapter 8).

Following the design-research paradigm, the contributions of this research can be analyzed from two aspects: (1) the development of theories or models that extend and improve the understanding of design problems, (2) and the development of methods or tools that enable solutions to the design problems (Chapter 9). More specifically, this study contributes to awareness support in collaborative activities at two levels:

1. First, we provide an integrated conceptual model for understanding the awareness phenomena in large-scale distributed collaborations. Our conceptual model is built on top of existing theories and models, and in turn contribute to existing knowledge foundations in two aspects: (1) We emphasize the distributed nature of the awareness phenomena, i.e. each team member's awareness is partial, but at the same time is compatible for the team to perform the collaborative activity successfully. (2) Our model provides a better explanation of how the compatibility of different collaborators' awareness is achieved through the integration of individual cognitive processes and social processes.

2. The second major contribution of this study is the computational awareness promotion approach that aims to design a knowledge-based awareness system that maintains a collective knowledge representation of the collaborative work, and utilizes it to support the various awareness processes. With the help of the formal knowledge representation, the awareness promotion approach shows several advantages to handle the scaled up complexity and dynamics in collaborative activities, and provides integrated awareness support.

Chapter 2

Understanding Awareness

This chapter attempts to identify the major issues for understanding awareness phenomena in large-scale distributed collaborative activities. A review of what is currently known about awareness at both individual and team levels is presented, which provides the grounding work for our conceptual model in next chapter.

2.1 Overview

The concept of awareness has come to play a central role in CSCW research. 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 CSCW research: *social* awareness and *activity-oriented* awareness [78, 93, 21].

Social awareness addresses the availability of different kinds of information related to the social context of 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" [57] and "a shared space for community building" [27]. Awareness of the general social context is an important aspect of collaborative work, especially in domains where cooperative work is defined in a loose and broad sense, or domains where socialization is crucial [93].

However, when the tasks of collaborators become closely interdependent on each other, more urgent concerns need to be given to the aspect of *activity-oriented* awareness. The *activity-oriented* awareness focuses on practices through which actors seamlessly align and integrate their distributed, 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 one's own work, of occurrence that makes one's work more urgent or leads to changes to the intended course of actions, etc. [93]. The major difference of *activity-oriented* awareness from *social* awareness is that it focuses on activities performed to achieve a specific shared goal [21] and the actor's being interdependent in their work [93].

In this study, we focus on the **activity-oriented** aspect of awareness in large-scale, distributed collaboration, which can be understood from two levels of analysis (Figure 2.1).

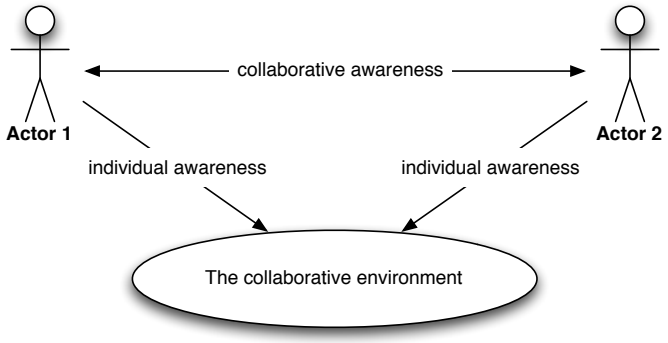


Figure 2.1. Two levels of analysis for understanding awareness

1. On one hand, because of the distributed nature of collaborative activities, the awareness is associated with individual actors, as each actor must be able to perceive the elements in the environment or related to other actors' actions, and then reason their consequences on his/her individual work. The awareness at individual level provides the basis for the team members to understand each other's work, and thereafter to achieve awareness at the team level. Existing studies on awareness at the individual level focus on how the awareness of an actor is achieved through interaction between a person and his/her environment.

2. On the other hand, there is a lot more to collaborative awareness than merely combining each team member's individual awareness [87]. To ensure that different actors' individual awareness is compatible with each other, and together results in a coordinated and complete team-level awareness of the collaborative activity, the actors have to interact with each other through team processes. Existing studies in collaborative awareness have focused on explaining how the team-level awareness is borne out of the interactions between individual actors.

The remainder of this chapter reviews existing studies related to these two levels of analysis respectively.

2.2 Individual 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 existing situation awareness models in the literature are individual focused theories [89], it has also been well recognized as an important element in collaborative environments. Gutwin and Greeenberg [50] 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. [21] also subsumes situation awareness with an emphasis on aspects of the situation that have consequences for group work towards shared goals. 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.

To understand the awareness at the individual level, two major aspects are well recognized in the literature, and are also applicable to collaborative environments: the *product* of awareness [33], i.e. what the awareness knowledge is composed of, and the *process* of gaining awareness [1], i.e. how awareness is achieved and developed.

2.2.1 The product of individual awareness

Among the numerous attempts to specify the product 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 levels [33]:

1. Level 1: *perception of relevant elements in the environment.* An actor must first be able to 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, an actor can forecast likely future states in the situation.

Endsley's three-level model presents an intuitive description of situation awareness at the individual level and has been applied in a plethora of different domains [115]. The division of SA into three levels allows the construct to be measured easily and effectively [30], and also supports the abstraction of 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 [50].

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 development of situation awareness [99, 112]. 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 individual awareness

To address the dynamic development of situation awareness, many researchers have used Niesser's perceptual cycle model [71] to clarify the cognitive components involved in the acquisition and development of situation awareness [99, 1, 50, 103]. According to the perceptual cycle model (Figure. 2.2), 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 actor's activity in the environment. That activity samples and perhaps anticipates or alters the environment, which in turn

informs the actor. The informed, directed sampling and/or anticipation capture the essence of behavioral characteristic of situation awareness.

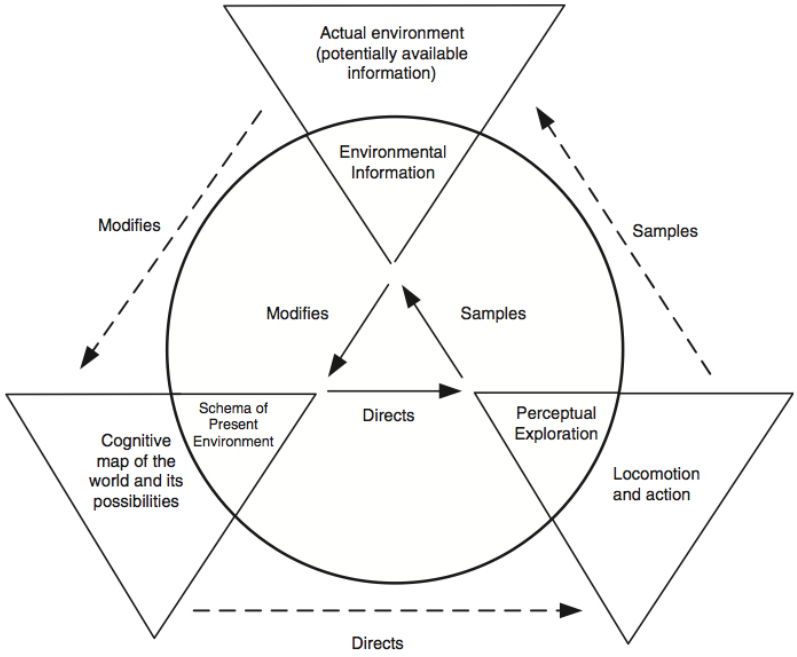


Figure 2.2. Niesser's perceptual cycle model [89]

Based upon Niesser's 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 [99]. Thus they view situation awareness as a generative process in "an adaptive cycle of knowledge, action and information" [99]. In a similar fashion, Adams et al. [1] used a modified version of Niesser's perceptual cycle model to describe how situation awareness works. They argue that the process of achieving and maintaining situation awareness revolves around an internally held mental model, which facilitates the anticipation of situational events, draw an actor's attention to cues in the environment and directs their eventual course of action. An actor then conducts checks to confirm that the evolving situation conforms to 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 [50].

One of the key assumptions of these models based on the perceptual cycle 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 suggested 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 [99], little detail has been given about the cognitive processes that guide the selection and interpretation of awareness information into the mental models.

In an attempt to clarify the cognitive processes involved in the development of individual awareness information, Bedny and Meister [7] propose a description of situation awareness based on the activity theory. They purport that individuals possess goals that represent an ideal image or desired end states of activities, which direct their actions to achieve 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.

Based on the activity theory, Bedny and Meister emphasize how the actor's goals and actions play a central role in the process of awareness development. Critical environmental features are identified based upon their significance to the individual's actions and the individual's motivation towards the task goal. The interpretation of these features in turn modifies an individual's goals and conceptual model of the current situation, which directs their activities and interaction with the world.

2.3 Collaborative awareness

Awareness in collaborative environments is indubitably more complex than awareness at the individual level. Beyond knowing what is going on in the environment, team members also need to develop an understanding of the actions of others to ensure that individual contributions are relevant to the group's shared goal as a whole [26]. Collaborative awareness is built on top of individual awareness, but at the same time requires the interaction between individuals to ensure that the combination of individual awareness

together is sufficient for the team to perform the shared activity.

The awareness at the team level can also be analyzed from the 'product' and the 'process' aspects. From the 'product' aspect, it is to understand the configuration of awareness knowledge in collaborative environments, i.e. how it is composed of the awareness of individual team members. From the 'process' aspect, it is to understand the interaction between team members through which the collaborative awareness is developed. This section reviews three prominent conceptualizations of awareness at the collaborative level, covering both the 'product' and the 'process' aspects (Table 2.1).

	The 'product' of awareness	**The 'process' of awareness**
Collaborative awareness as shared knowledge	individual team member's awareness and the shared awareness knowledge with other team members	effective team processes for sharing relevant information, such as communication and mutual monitoring
Collaborative awareness as shared activities	the shared context surrounding a collaborative activity, e.g. the shared goals and status, the top-down goal decomposition, the dependencies within the actions, etc.	the joint construction of common ground, shared practices, social capital, and human development
Collaborative awareness as distributed cognition	distributed individual awareness that is overlapping, and complementary with each other	transactions representing the exchanging of awareness between actors

Table 2.1. Conceptualizations of collaborative awareness

2.3.1 Collaborative awareness as shared knowledge

The research on team situation awareness (TSA) attempts to directly extend the theories and models of situation awareness to collaborative settings. TSA conceives the collaborative awareness as a 'shared understanding' of the same situation. 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 roles in the team, but also elements that are shared with other members in the team. It is therefore argued that, at a simple level, the collaborative awareness comprises two separate but related components: team member's individual awareness, and the shared awareness with other team members (Figure 2.3).

The team situation awareness adopts the knowledge-in-common view [69], i.e. it focuses on how the shared understanding of the situation is developed at the team level. Nofi [72], for example, defines team SA as: "a shared awareness of a particular situation" and Perla et al. [76] suggest that "shared SA implies that we all understand a given

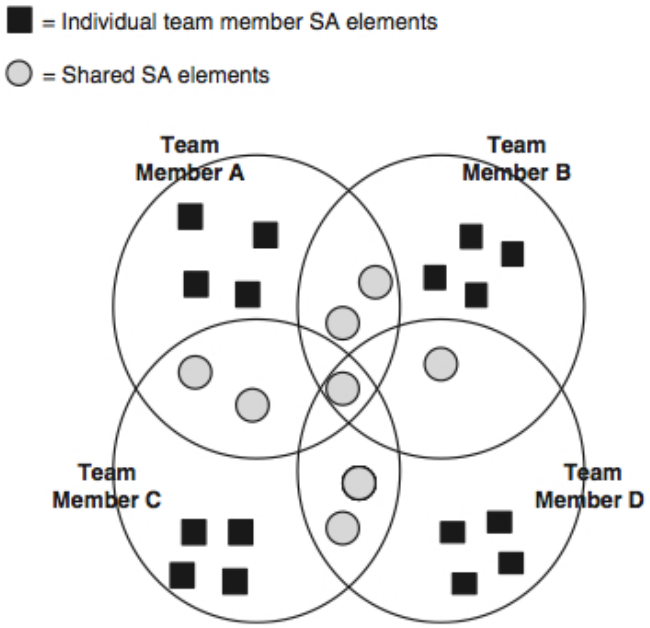


Figure 2.3. Shared situation awareness (adapted from Endsley 1995 [33])

situation in the same way". Shu and Furuta [96] point out 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". As a result, a critical factor of team situation awareness is to define the configuration of shared awareness requirements, i.e. to identify the awareness knowledge that needs to be shared with the team.

By considering the collaborative awareness as shared knowledge among team members, the development of awareness at the team level relies on effective team processes for sharing relevant information. Salas et al. [87] propose a framework of TSA that comprises two critical processes, individual situation awareness and team processes. The development of collaborative awareness has a cyclical nature of developing individual SA, sharing SA with other team members, and then modifying SA based on other team members' SA. Most researchers have focused on communication as the key team process in the development of collaborative awareness. Nofi [72], for example, cites communication as the most critical element in the creation of shared SA . Salas et al. [87] argue that team members acquire individual awareness, and then communicate this throughout the team, which leads to a common team understanding. Entin and Entin [34] note that communication is a prerequisite for high levels of team SA. Another key team process

that is critical to team SA is the process of mutual monitoring, whereby team members monitor one another's activities, allowing the sharing of awareness information without explicit verbal communication [50, 89]. Although 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].

The heavy emphasis of collaborative awareness on shared knowledge is simplistic in large-scale distributed collaboration. As argued by Mohammed and Dumville [69], the knowledge-in-common view may be appropriate for only certain task domains and types of groups. 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. As we characterize the collaboration as highly distributed where collaborators play specialized roles or attain distinct knowledge in the course of joint activities, they experience the situation in different ways. 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].

2.3.2 Collaborative awareness as shared activities

To address the problem of the 'shared knowledge' view of collaborative awareness, Carroll et al. proposes a new framework for understanding awareness in collaborative environment, based on the concept of 'activity awareness' [21, 22]. The major distinction between TSA 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 [22]. In framing activity awareness, they appropriate the concept of *activity* from Activity Theory to emphasize that collaborators need to be aware of a shared activity as a complex, socially embedded endeavor, organized in dynamic hierarchies, and not merely aware of the synchronous and easily noticeable aspects of the activity [23].

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 product and process 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, i.e. 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 [21].

By shifting the focus from shared knowledge to shared activities, activity awareness aligns the development of awareness with the development of collaborative activities. Most basically, activity awareness is achieved and developed through the joint construction of common ground - shared knowledge and beliefs, mutually identified and agreed upon by members through a rich variety of communication protocols [22]. 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 develop team member's activity awareness.

Activity awareness with its basis on Activity Theory, provides a level of abstraction that is more constructive and dynamic to structure the sharing requirement in collaborative awareness. 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 [103]. 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 detail about how the activity knowledge is distributed across multiple actors.

2.3.3 Collaborative awareness as distributed cognition

A more recent theme to conceptualize awareness in collaboration is the concept of distributed or systemic team awareness [103, 6]. Distributed team awareness approaches are borne out of the 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 assumes that awareness is distributed across different human and technological

agents involved in collaborative systems [103].

The main difference between distributed team awareness and other TSA and activity awareness models is related to the concept of *compatible* awareness [103]. Distributed team awareness postulates that, within collaborative systems, each team member does not need to know everything, rather they possess the awareness that is needed for their specific tasks. Although different team members may be aware of the same information, this awareness is not shared, since the team members often have different goals and so view the situation differently based on their own tasks and goals. Different team member's individual awareness can be different in content but at the same time is compatible in that it is all collectively needed for the overall team to perform the collaborative activity successfully [90]. The team members' individual awareness can be overlapping, and complementary with each other, and hence deficiencies in one actor's individual awareness can be compensated by another actor. To use the analogy of a cog in a machine, each cog does not need to know about all the other cogs, rather it needs only to be able to connect with those cogs adjacent to it (Figure 2.4). Thus it is suggested that 'compatibility' is the key to collaborative awareness, rather than 'sharedness' [88].

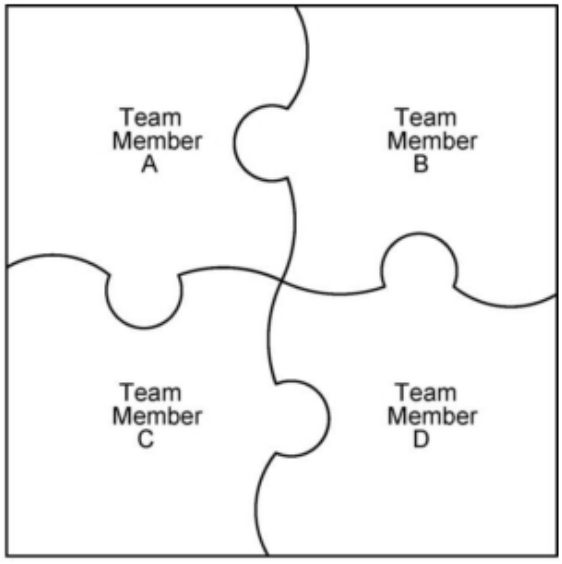


Figure 2.4. Compatible awareness (adapted from Salmon et al. [103])

While the distributed team awareness emphasizes the distribution of awareness, it does not discount the interactions among different team members. The 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 knowledge between actors. Actors receive information, integrate it with existing knowledge, and then pass on to other agents. The interpretation on that information changes per team member. Transactions between team members lead to awareness knowledge being passed around. For example, an actor may perceive certain awareness element in the environment, interpret the meaning, and then pass it to another actor via a transaction. The second actor then builds its own interpretation upon the first actor's interpretation, and may start a new transaction to pass his/her understanding to other actors. Hence, it is the systemic transformation of awareness elements as they cross the local boundary from one team member to another that bestows upon awareness in collaboration an emergent behavior [103].

The concept of distributed team awareness has been investigated in a number of domains, including naval warfare [102], energy distribution [88], and air traffic control [103]. 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 [103]. However, the main weakness, as admitted by the authors, is 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 in a collaborative environment. A possible remedy is to integrate the distributed team awareness with the activity-directed models and switch the basic unit of analysis from arbitrary concepts to activities to understand the awareness phenomena.

2.4 Discussion

By reviewing the existing theories and conceptual models of awareness, we believe that none of them alone can account for all the aspects of awareness phenomena in large-scale distributed collaboration:

1. The models of situation awareness primarily focus on the individual level of awareness phenomena, and have limited support for collaborative awareness.

2. We agree with the distributed team awareness approaches [90] on that, the conceptualization of collaborative awareness merely based on the idea of 'sharing' is an oversimplification in large-scale distributed collaboration. As we characterize the collaboration as highly distributed where collaborators play specialized roles or attain distinct knowledge in the course of joint activities, the interaction be-

tween the actors to achieve collaborative awareness is more about transactions than information sharing.

3. We resonate with the activity directed SA model [7] and the activity awareness framework [21] that existing models using concepts as the basic unit to analyze awareness elements tend to be too static and rigid for collaboration with higher level of dynamics.

As a result, instead of adopting one particular viewpoint, we believe that a suitable conceptual model for understanding awareness in large-scale distributed collaborative activities should integrate multiple models in the literature. Specifically, we identify the following requirements for such a conceptual model of awareness:

Integration of individual and collaborative awareness The conceptual model should be able to account for awareness at both the individual and the team levels, and emphasize how these two aspects interplay with each other. On one hand, because of the distributed nature of these collaborative activities, awareness is associated with individual actors, as each actor must be able to achieve the individual awareness that provides the basis for them to understand each other's work. On the other hand, collaborative awareness requires the actors to interact with each other through team processes so that their individual awareness can be meshed up together.

Support for distributed awareness Because of the differences in goals, roles, and tasks, each team member's awareness is different in content, but at the same time is compatible for the team to perform the collaborative activity successfully. Hence, the conceptual model should be able to account for how the awareness is distributed across multiple team members, and how they interact with each other to achieve compatibility.

Integration of awareness model and activity model The conceptual model should emphasizes the importance of using the concept of activity to structure the product and process of awareness phenomena. As pointed out by Schmidt [93], when we are talking about 'awareness', we are talking about the phenomena that actors align and integrate their actions with the actions of others to achieve their shared goals. As a result, any need for the actors to acquire and maintain awareness in the collaborative environment arises out of their need to perform their activities and not for its own sake.

Chapter 3

An Integrated Conceptual Model of Awareness

In this chapter, we propose an integrated conceptual model of awareness in large-scale distributed collaborative activities that can satisfy the general requirements identified in Section 7.4:

1. The conceptual model should be able to account for awareness at both the individual and the team levels (*REQ1*).
2. It should be able to account for how the awareness is distributed across multiple team members, and meanwhile can interact with each other to achieve compatibility (*REQ2*).
3. Awareness should be described in reference to activities that the actors are made aware of (*REQ3*).

To satisfy *REQ3*, we build the model of awareness on top of an underlying model of collaborative activities. The activity model plays two important roles to understand the awareness phenomena. First, the three constructs of the activity model, i.e. activity, local scope, and dependency, provide the basis to explain the distributed and dynamic nature of the awareness phenomena (*REQ2*). Second, the activity model is used to understand the development of awareness at both the individual and team levels (*REQ1*). At the individual level, it is used to explain the various cognitive processes that are associated with an individual to achieve and develop individual awareness. At the collaborative level, it accounts for the team processes through which team members interact

with each other and mesh their individual awareness together to achieve collaborative awareness.

In the following of this chapter, we provide the details of our conceptual model in three steps: we first describe the underlying model of collaborative activities in Section 3.1; then we use the activity model to characterize the awareness phenomena in large-scale distributed collaborative activities (Section 3.2); last we describe the awareness processes through which the collaborative awareness is developed (Section 3.3).

3.1 A model of collaborative activities

Our model of large-scale distributed collaborative activities is built on top of three interrelated constructs (Figure 3.1):

1. We appropriate the concept of *activity* from Activity Theory [70] to structure the collaborative work.

2. Due to the distributed nature of collaborative activities, each team member only engages in a subset of actions in the collaborative work, and forms their respective *local scopes of work*.

3. Actions in different local scopes of work are interdependent due to various types of *dependencies* among them.

In general, a collaborative activity can be defined as a domain model that includes three related sub-models $CoA = \{ER, LS, DEP\}$: the structure of the activity composed 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 these three components in the rest of this section.

3.1.1 Structure of a collaborative activity

To model collaborative activities, we subscribe to Activity Theory to conceptualize human activities [70]. From this perspective, the structure of a collaborative activity can be defined as several basic entities and relations between them.

3.1.1.1 Entities

We define three entities that are the basic building blocks in a collaborative activity:

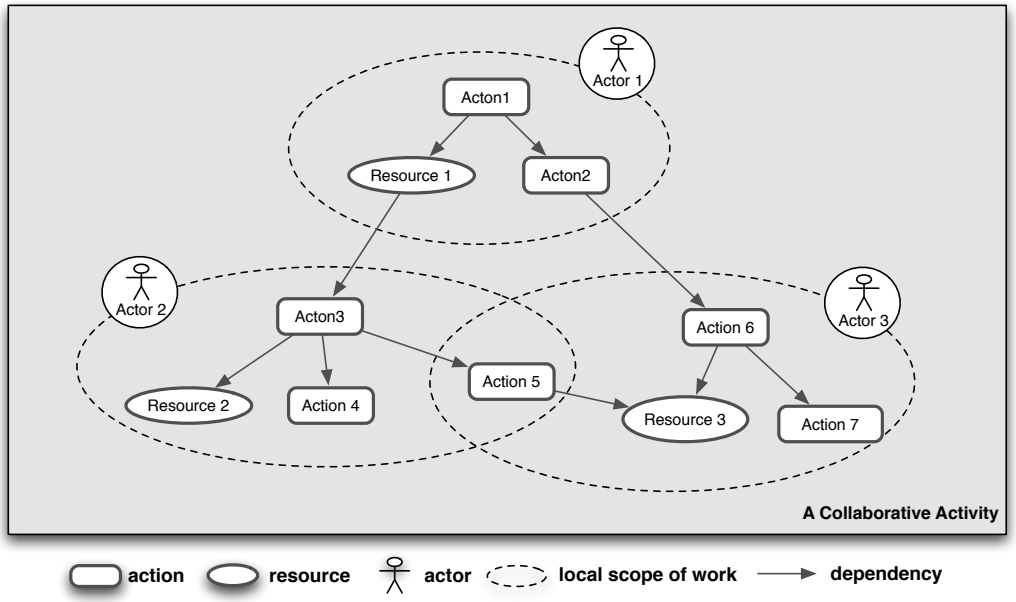


Figure 3.1. A model of collaborative activities

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 one or more actors who are responsible to perform it. We use $ACT = \{act_1, act_2, ..., act_n\}$ to denote all the actors in a collaborative activity.

2. **Actors** are defined as *subjects* intend to and have the capability to perform actions. Actors cast their mental attitudes towards the corresponding action and these internal states are important for the success of an action. $AR = \{ar_1, ar_2, ..., ar_n\}$ is the set of all the actors in a collaborative activity.

3. **Resources** are anything that can be used in the performance of an action. One one hand, they can be the *objects* that are manipulated or transformed by the actors while performing actions. On the other hand, they can be the various *tools*, such as instruments, procedures, machines, etc, that are used by the actors to perform their actions. The concepts of object and tool in a collaborative activity are often relative. A tool may be used by an actor in the transformation process of an object, and at the same time the tool itself can be created or transformed

by another actor, where it becomes an object of the other actor's action. As a result, we use *resource* as a general term for both *objects* and *tools*. Resources can be can be a material thing (e.g. a vehicle), or intangible (like a rescue plan). $RES = \{res_1, res_2, ..., res_n\}$ denotes the set of resources.

3.1.1.2 Relations

The various entities in a collaborative activity are not standalone, rather are connected together to achieve the shared goal of the activity. The relations between these entities can be defined using predicate symbols. For example, predicate symbol $LocatedIn$ 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 $LocatedIn(ar_1, room_3)$. 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 structures, and are used later to define local scopes of work LS and dependencies DEP.

Relations between action and actor The relations between an action and an actor reflect the levels of participation of the actor in the performance of the action, which can be defined from two aspects: the *intention* of the actor to the action performance, and the *capability* of the actor to perform it. One one hand, the actor participates in an action because of the 'motive' or 'goal' that arises out of the collaborative activity, and drives the actor to participate in the performance of the action. On the other hand, to what extent the actor can contribute to the action depends on the actor's capability in relation to the action performance.

We follow the SharedPlans theory [47] to define three levels of intentions towards an action that an actor can adopt:

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

2. $Int.Th$ represents an actor's intention that the action should be performed.

3. $Int.To$ is used to express an actor's intention to perform the 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 act_1 should be performed, but has not yet committed to it.

$Int.Th(ar_1, act_1)$ indicates that ar_1 has intended that act_1 should be successfully performed. In this case, ar_1 may help other actors to get it done, or avoid resource conflicts, but does not directly perform act_1. In the case of $Int.To(ar_1, act_1)$, ar_1 has full commitment to perform act_1, i.e. ar_1 directly participates in the performance of act_1.

The *capability* of an actor to perform an action can also be defined at three different levels:

1. $Knows$ is used to indicate that the actor has the knowledge about how to perform the action, i.e. the actor knows how to decompose the action into subsidiary actions or basic operations.

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

3. $Workable$ indicates that the actor can actually perform the action as all the preconditions have been satisfied, such as the availability of required resources.

These three relations 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 act_1, but may not be able to perform it. For example, a blind man may know how to drive a car, but is unable to drive it. $Able(ar_1, act_1)$ indicates that ar_1 has the ability to perform act_1, but it does not necessarily mean the actor can actually perform it. A man who is able to drive the car does not have the key, and hence he cannot actually drive it. $Workable(ar_1, act_1)$ defines the highest level of capability, as ar_1 can meet all the constraints and actually perform 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, has 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. *Sub.Act* only indicates the relation 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, the actions can also be related by the *precedence* relation. *Precedes* defines the temporal order of performing two actions, i.e. $Precedes(act_1, act_2)$ indicates that act_1 must be performed before act_2.

Relations between action and resource The relations between an action and a resource can be identified depending on the role of the resource in the action performance. One one hand, the resource can be a *tool* that is used in the performance of an action. On the other hand, the resource can be the *object* that is manipulated or transformed during the action performance. 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 (as a *tool*). $Produces(act_1, res_1)$ indicates that the performance of act_1 will manipulate the resource res_1, or make the resource ready for other actions (as an *object*).

3.1.2 Local scope of work

Within a large-scale collaborative environment, a good number of actions can be identified. However, Each actor usually only engages in a small set of them. In fact, one of the fundamental goals for collaboration is to divide the labor so that a complex problem can be decoupled into a set of smaller ones that are much easier to manage and tackle by individuals [92]. One result of the division of labor is that the collaborative work of is distributed in the sense that each actor only needs to work on a small set of actions that are suitable for their roles, skills, and experience. To characterize the distributed nature of collaborative activities, we define the **local scope of work** for each actor as the set of actions that the actor participates in.

Because the participation of an actor in an action can be defined using the *intention* and *capability* relations (Section 3.1.1.2), the local scope of work for an actor can be identified as follow.

1. The *intention* relations are used to define the set of actions that an actor has intention towards their performance. This set of actions form the sub-space of the collaborative activity that the actor is willing to work on, we call it *local scope of intention*. Formally, the *local scope of intention* for an actor $LSI(ar)$ is defined 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 $rel \in \{Pos.Int, Int.Th, Int.To, Perform\}$.

2. The *capability* relations define the set of actions that the actor have certain level of capability to work on. This set of actions form the sub-space of the collaborative

activity that the actor can work on, we call it *local scope of capability*. 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 $rel \in \{Knows, Able, Workable\}$.

The whole *local scope of work* for an actor is then defined as the union of these two sub-spaces defined by the *intention* and *capability* relations (Figure 3.2):

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

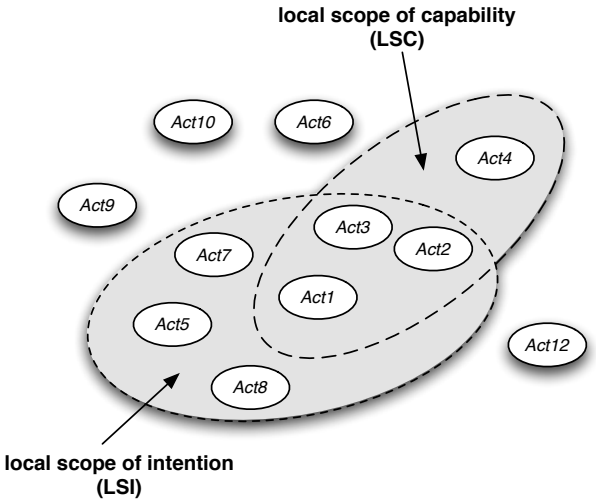


Figure 3.2. Local scope of work

The actions within an actor's LS can have different implications on the actor. The actions within the intersect of these two sub-spaces (e.g. $Act1, Act2, Act3$ in Figure 3.2) are these that the actor actively participates in because the actor both has certain level of intention and capability towards their performance. The actions within LSI, but outside LSC (e.g. $Act5, Act7, Act8$) are these that the actor may need help from other actors as the actor intends that they should be performed, but does not have the necessary capability to do it. The actions outside LSI, but inside LSC (e.g. $Act4$) are where the actor can offer help to other actors, even though the action is not initially intended by the actor.

An important property of the *local scope of work* is that local scopes of different actors can overlap with each other. It is common that one action falls into local scopes

of multiple actors, even though it may have different implications on these actors. Multiple actors may intend to perform the same action together as a subgroup. The action that is initially intended by one actor may actually be performed by another actor. The different actors may contribute to the same action differently, as one actor contributes the knowledge about how to decompose it in to sub-actions, and the other actor then performs these sub-actions. In all these situations, there are some actions reside in multiple actors' local scopes of work. Actually, these shared actions across overlapping local scopes of work play an important role in understanding the compatibility of awareness that we will discuss in next section.

3.1.3 Dependency

Although actions are largely distributed and belong to local scopes of different actors, they cannot be performed without interacting with each other. The *dependencies* between actions represent the knowledge about why and how the actions depend on each other towards the accomplishment of the overall collaborative activity. While local scopes of work divide the actions in a collaborative activity into multiple parts so that each actor can only work on a small portion of it, dependencies show how distinct actions in different local scopes can still be related to each other.

In general, a dependency is defined as a meta-predicate (DEP) on two actions (act_1, act_2) and a proposition p, where the performance of act_1 depends on the performance of 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 that 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, we can model three types of dependencies that have been recognized in the literature: temporal dependencies, resource dependencies, and goal dependencies.

Temporal dependencies Different actions in a collaborative activity might be interdependent due to the constraint of ordering them in a certain order [97]. The predicate $Precedes$ can be used to define the temporal dependency between actions, i.e. the performance of act_2 as *depender* depends on the performance of 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 Resource related dependencies can be analyzed in terms of common resources that are involved in multiple actions. Different patterns of use of the common resources by the actions will result in different kinds of resource dependencies. Malone et al. [63] classify three types of resource dependencies as shown in Figure 3.3, which can be defined using the predicates *Consumes* and *Produces*:

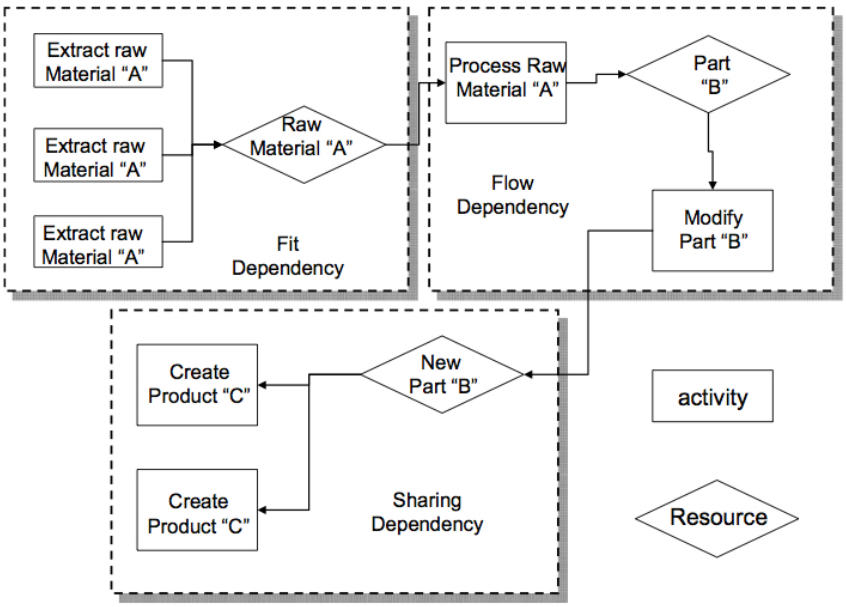


Figure 3.3. Types of resource dependencies [63]

1. 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 $Produces(act_2, res_1)$.

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

2. 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 $Consumes(act_2, res_1)$.

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

3. A *sharing dependency* arises whenever two actions 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 dependencies A goal dependency reflects the fact that one action depends on the other action to bring about a certain state in the world. Goal dependency is usually characterized by the decomposition relation between two actions, i.e. action A depends on the other action B because B is a means to achieve a subsidiary goal that must be satisfied in order to perform A.

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 activity, i.e. $Sub.Act(act_1, act_2)$

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

3.1.4 Development of a collaborative activity

As argued in Activity Theory [70], the collaborative activities are not static or rigid, but rather they and their elements are under continuous change and development. This becomes even more significant in large-scale distributed collaborative activities where the actors work in dynamic settings that entail frequent changes in the environment. As argued by Carroll et al. [22], in these open-ended collaborative activities, the 'normal' plans of shared activities cannot be precisely specified in advance, and actions and roles cannot be rigidly a priori. As a result, the model of collaborative activities should incorporates several important developmental trajectories:

1. Goal elaboration [48]. The development of a collaborative activity usually involves the top-down goal decomposition in which the shared activity is elaborated into

multiple actions to complete it.

2. Opportunistic plan revisions [105]. Even after the plan of the shared activity has been identified, some part of the plan may have to be revisited and revised because of the changes in the environment.

3. Role transfer [111]. In some dynamic situations, it is impossible to predict who will undertake what specific role. Actor may have to perform actions that are outside their initial responsibilities.

Because of these developmental trajectories that may occur in a collaborative activity, all the constructs in our model of the collaborative activity can be under continuous change.

1. First, the basic entities and rations in the collaborative activity may be changed. New actions or resources may be added to the model as the plan is developed or revised. The relations between actions and resources may be changed because the same resource is assigned to different actions at different stages of the collaboration.

2. Second, the local scopes of work are also dynamic. The local scope of an actor may be expanded as the actor elaborates on the plan of an action, or it can include different sets of actions as the role of the actor is transferred.

3. Last, the patterns of dependencies among actions are also quite volatile. While two actions may stay the same during the development, new dependency relation may be cast on them because of the possibility of re-planning.

3.2 Characteristics of awareness

The purpose of modeling collaborative activities in previous section is to structure the awareness phenomena on top of it, based on the assumption that any need for the actors to acquire and maintain awareness in a collaborative activity arises out of their need to perform actions [93]. That is to say, what actors are supposedly aware of depends on how they are acting in the collaborative activity. The alignment of awareness phenomena and the underlying activity model allows us to identify three important characteristics of awareness in large-scale distributed collaboration.

3.2.1 Partiality of individual awareness

The first characteristic of awareness is based on the concept of local scope of work. The idea is that, since each actor only engages in a small set of actions in a large-scale collaborative environment, the actor does not need to fully understand every detail of the whole situation. Instead, the information that an actor should be aware of is only partial, depending on whether it is related to the actions within the actor's local scope of work. An actor would be aware of the information that ascertains or changes the state, progress, direction of one's own work; of the occurrences that makes one's own work more urgent or less; of things that necessitate changes to the intended course of actions to mesh with the unfolding work of others, etc [93]. All of these awareness requirements are motivated and constrained by the actor's local scope of work.

Figure 3.4 illustrates the idea of partiality of individual awareness in a collaborative activity, where each actor only needs to take heed of a partial set of awareness elements that arise out of the actor's corresponding local scope of work. Even though one action can belong to multiple actors' local scopes, the actors may be interested in different aspects of the same action and hence each actor's individual awareness is unique and used for their own work.

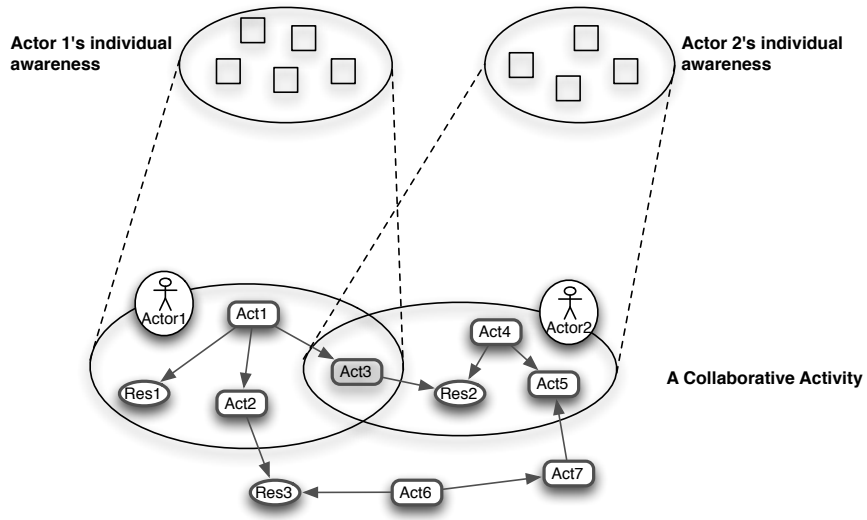


Figure 3.4. Partiality of individual awareness

The partiality of individual awareness is related to existing theories about local reasoning [8, 45], which argue that a human actor's reasoning is often based on 'local' facts that form the individual's cognitive context [8]. Such a context is local because it usually

only includes information that is potentially relevant to the individual's tasks at hand. Hence, a significant advantage of partitioning the awareness space based on the local scopes of work is that it makes each actor's reasoning more efficient by reducing the number of potential knowledge elements that need to be taken into account [45].

3.2.2 Compatibility of awareness

However, because of the relations among different actors' local scopes of work, the awareness phenomena in large-scale distributed collaboration is not simply equivalent to the partition of the collaborative awareness space into separate individual awareness spaces. Rather, the actors' individual awareness needs to be coordinated so as to achieve the *compatibility* that is collectively needed for the overall team to perform the collaborative task successfully. The requirement for compatible awareness among multiple actors can be accounted for by two typical patterns in the activity model: the overlaps between local scopes of work, and the dependencies across local scopes (Figure 3.5).

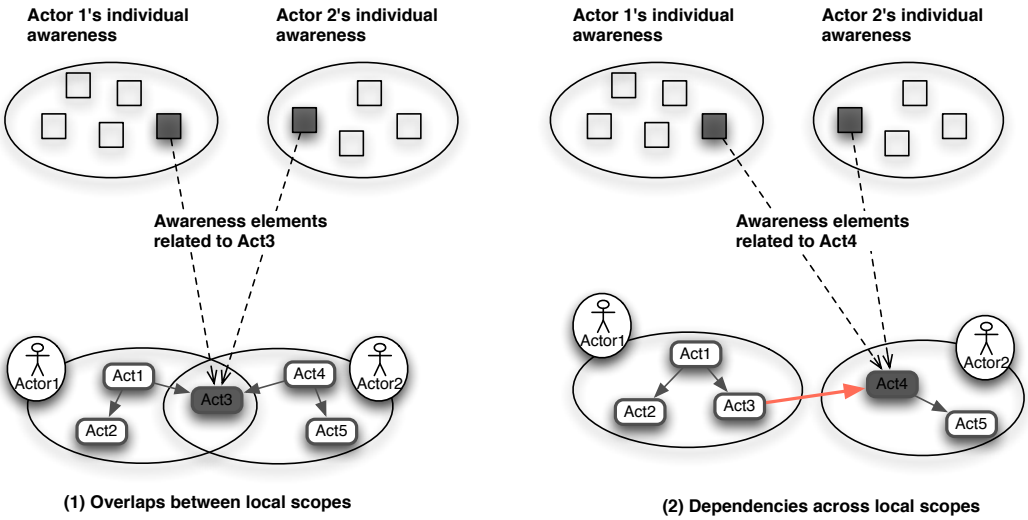


Figure 3.5. Compatibility of awareness

Overlaps between local scopes Although the actions in different actors' local scopes of work can be quite different, it is quite common that some actions sit in between local scopes of different actors as boundary objects. Multiple actors may work as a subgroup to perform the same action together, or an actor delegates an intended action to another actor who has the capability to perform it. These boundary objects pose the need for the actors to achieve compatibility. The actors working on the same action need to share

their individual interpretations of the action with each other to make sure they are on the same page. The actor who is performing the delegated action needs to update the status to the actor who adopts the initial intention so that he/she knows what to expect.

Dependencies across local scopes The second driving force to achieve compatible awareness comes from dependencies that can connect actions from two disjoint local scopes of work. For example, in Figure 3.5(b), because the action $Act3$ of $Actor1$ depends on $Actor2$'s action $Act4$, $Actor1$ will also be interested in knowing the information related to $Act4$, such as the fact that $Act4$ has been successfully completed or it encounters problems. In such case, even though $Act4$ is outside $Actor1$'s local scope, $Actor1$ still wants to be aware of aspects of $Act4$ that are likely to impact his/her own action $Act3$ due to the dependency relation between these two actions.

The compatibility of awareness between actors is achieved by awareness *transactions* [90]. An awareness transaction represents an exchange of individual awareness between two actors. One actor receives information and interprets it. If the result of the interpretation is related to the shared action with the other actor, or it is a dependee of the other actor's action, it is passed to the other actor. In this way, the compatibility between the two actors is achieved. In next section, we will provide more detail about how awareness transactions can be achieved through team processes.

3.2.3 Dynamics of awareness

As we pointed out in Section 3.1.4, the elements and relations, local scopes of work, and dependencies are all under continuous change and development in large-scale distributed collaborative activities. As a result, we can postulate that an actor's awareness requirement, i.e. the information an actor need to be aware of, also entails frequent changes. The awareness elements that were relevant to the actor may later become irrelevant, as they are related to the action that is no longer part of the actor's local scope of work. New awareness elements may have to be added to the actor's awareness requirement as the local scope expands or new dependency is activated. Figure 3.6 shows an example of the dynamic of awareness as an actor's local scope of work is changed due to plan development.

The dynamics of awareness is coupled with the development of the collaborative activity. As we will show in next section, the collaborative awareness and activity are mutually developed through a set of individual and team processes. The actors' awareness requirements are changed as the collaborative activity is developed. Guided by the

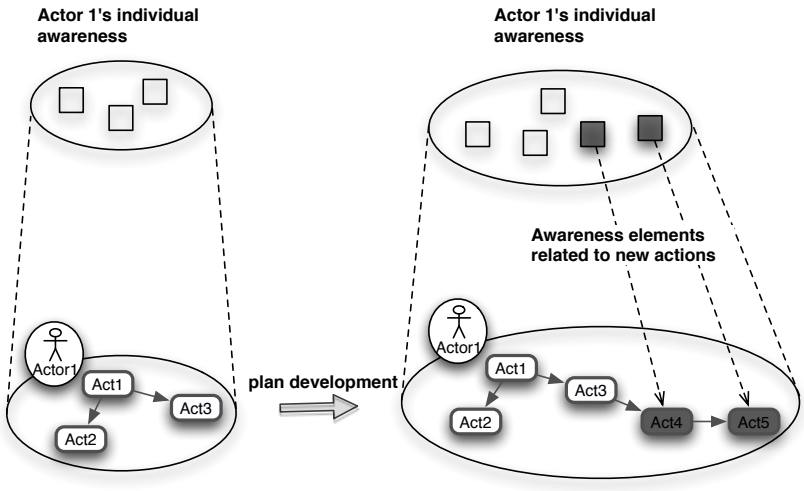


Figure 3.6. Dynamics of awareness

new awareness requirements, the actors process new awareness information and adjust their behaviors that lead to further development of the collaborative activity. This is very similar to Niesser's perceptual cycle model [71], but applies to the collaborative environments.

3.3 Awareness processes

Because of the partiality and compatibility of awareness in large-scale distributed collaboration, we believe that awareness is developed at both the individual and team levels. At the individual level, each actor develops his/her own partial awareness through a set of cognitive processes. At the team level, the actors engage in team processes to perform transactions in order to achieve compatible awareness.

3.3.1 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 [71]: the development of awareness starts with the perception of selective elements, which is then interpreted with the help of the actor's existing knowledge. The result of the individual awareness processes then helps the actor to make decisions and perform actions, which generate new awareness elements and trigger another round of awareness processes (Figure 3.7).

The major difference in our model from the original perceptual cycle model is that the

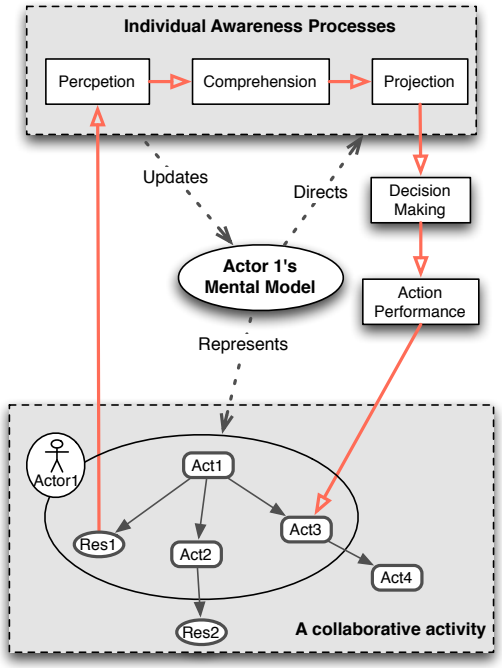


Figure 3.7. Development of individual awareness

actor's local scope of work plays important roles in the individual awareness processes:

1. *Perception.* An actor must first be able to gather perceptual information in the surrounding situation, and selectively attend to those elements that are most relevant for the task at hand. The perception is constrained by the actor's local scope, i.e. he/she is only interested in awareness information that have explicit or implicit impact on actions 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 actions in the local scope, i.e. the actor needs to understand how the perceived information is related to his/her own work.

3. *Projection.* Projection is the process in which the actor predicts likely future states of the actions in the local scope of work.

3.3.2 Development of collaborative awareness

The development of awareness at the team level is mediated by the transactions between multiple actors. Transactions allow for the exchange of individual awareness between ac-

tors to achieve the compatibility. The development of awareness at the team level usually entails multiple transactions through which actors' individual awareness processes are connected as developmental trajectories.

3.3.2.1 Awareness transactions

An awareness transaction represents an exchange of awareness information between two actors to achieve compatibility. Transactions happen whenever compatibility of individual awareness becomes necessary. If the result of one actor's individual awareness is related to the shared action with the other actor, or is related to the action that the other actor's action depends on, it is exchanged with the other actor through an awareness transaction. We call the first actor the *initiator* of the transaction, and the second actor the *receiver* of the transaction.

Awareness transactions can be achieved by joint effort from both the initiators and receivers through two complementary awareness practices: monitoring and displaying [52]. One one hand is the initiators display their own actions appropriately visible to other actors, and at the same time the receivers actively monitor the actions of other actors in the collaborative setting. Based on how the effort is divided between initiators and receivers, we can identify three types of awareness transactions.

Mutual monitoring The first type of awareness transactions refers to the process of actors mutually monitoring the actions of each other in a collaborative environment. The perception of such actions allows the actors to understand how other actors' actions can impact their own actions so as to achieve compatibility. An awareness transaction based on mutual monitoring directly connects two actors' individual awareness processes in the following way (Figure 3.8). First, some awareness element in the situation triggers the first actor ($Actor1$)'s individual awareness process, in which $Actor1$ perceives the awareness information, comprehends and projects it. The result of the actor's individual awareness process leads to some action performed by $Actor1$. The performance of the action is then perceived by the other actor ($Actor2$), as $Actor2$ is actively monitoring it. As a result, $Actor2$ starts the individual awareness process to consume the perceived information. In this way, the actors do not need to engage in any explicit team processes, as the transaction is implicitly achieved by the actors' individual perception.

Awareness transactions via mutual monitoring primarily rely on the receivers to perceive the awareness information from the field of collaborative work and assume that the aspects of actors' actions be visible to each other. However, it has two limitations.

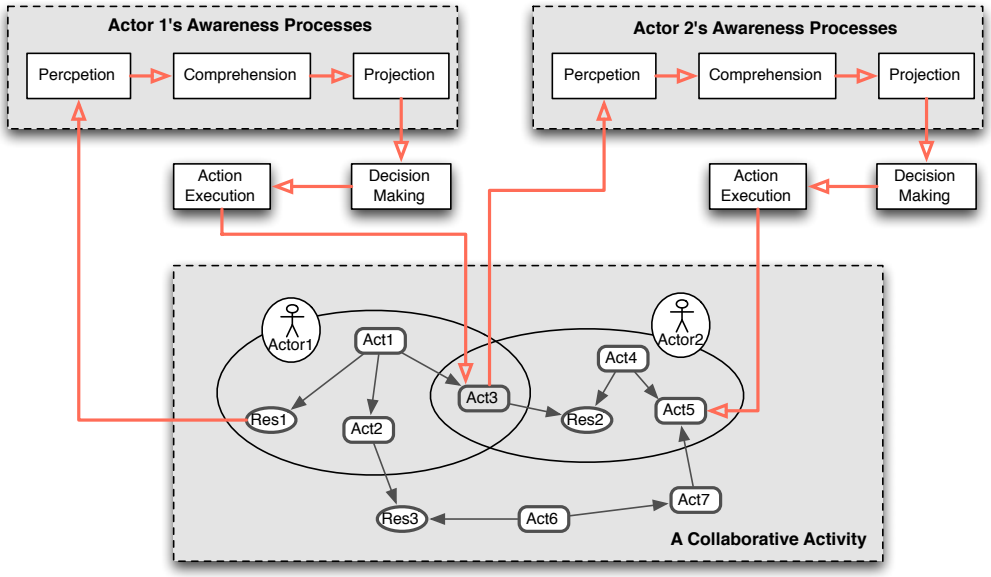


Figure 3.8. Awareness transaction via mutual monitoring

1. Without explicit externalization or disclosure, the actors can only monitor external aspects of other actors' actions [83]. However, many aspects of human actions are intentional in the sense that the awareness information related to these actions is the state of someone else's mind [21]. As a result, the awareness information on these aspects cannot be exchanged merely based on mutual monitoring.

2. In order to perform the awareness transactions via mutual monitoring, every actor in the collaborative activity needs to spare some attentional resource to each other's actions. When the number of actors and their actions grows significantly, it requires a lot of extra attentions and efforts from the actors.

Communication Alternatively, awareness transactions can be conducted via intentional communications, in which actors explicitly exchange awareness information with their collaborators through direct communications. As shown in Figure 3.9, the first actor ($Actor1$) can explicitly state the result of his/her individual awareness processes, and convey it to the other actor ($Actor2$) who might be interested in knowing it. Upon receiving the communicated information, $Actor2$ starts the individual awareness process to consume it.

Unlike awareness transactions via mutual monitoring, the transaction cost in the communication process is largely posed on the actor who initiates it. The initiator needs

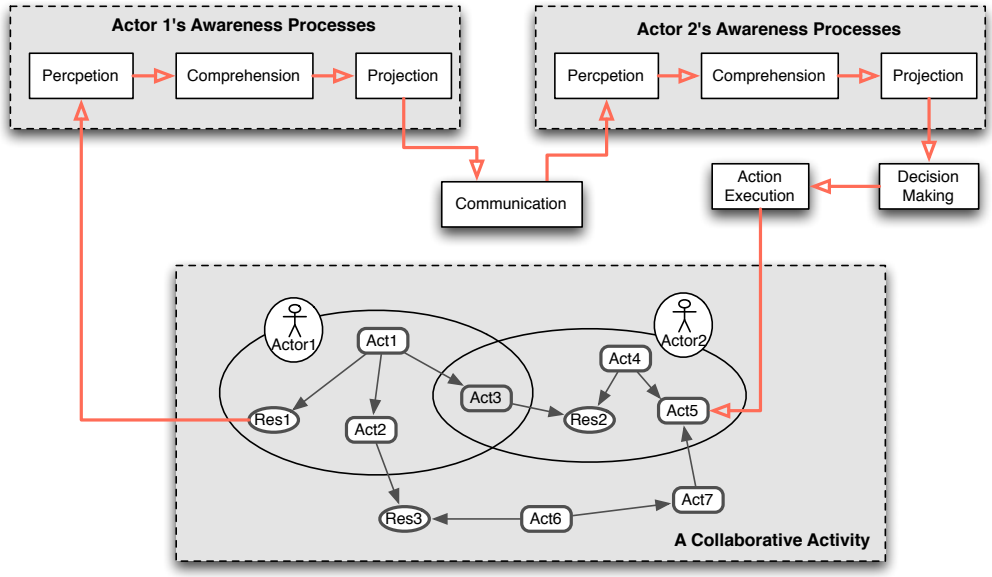


Figure 3.9. Awareness transaction via communication

to first create the message that he/she wants to communicate, identifies the collaborators who might be interested in knowing it, and starts the communication. Besides, the communication may be too obtrusive that it causes interruptions on the receivers, as they have to switch their focus from the current line of work in order to consume the information.

Externalization Externalization refers to an indirect means to exchanging intentional aspects of awareness among team members. Instead of explicit communication, team members can externalize some intentional aspects of their individual awareness and make them visible to others, so that anyone who is interested in these aspects can perceive them. As shown in Figure 3.10, *Actor*1 first externalizes the result of his/her individual awareness processes on an action. Instead of communicating it directly to *Actor*2, *Actor*1 attaches it with the action and makes it visible to others. The externalized information is then perceived by *Actor*2, and *Actor*2 starts the individual awareness process to consume the information.

The awareness transactions via externalization balance the efforts between the initiators and the receivers. On one hand, the initiators need to explicitly externalize some intentional aspect of their individual awareness and attach the externalized information with related actions. On the other hand, the actors who are interested in receiving information about these actions need to actively monitor them so that the information

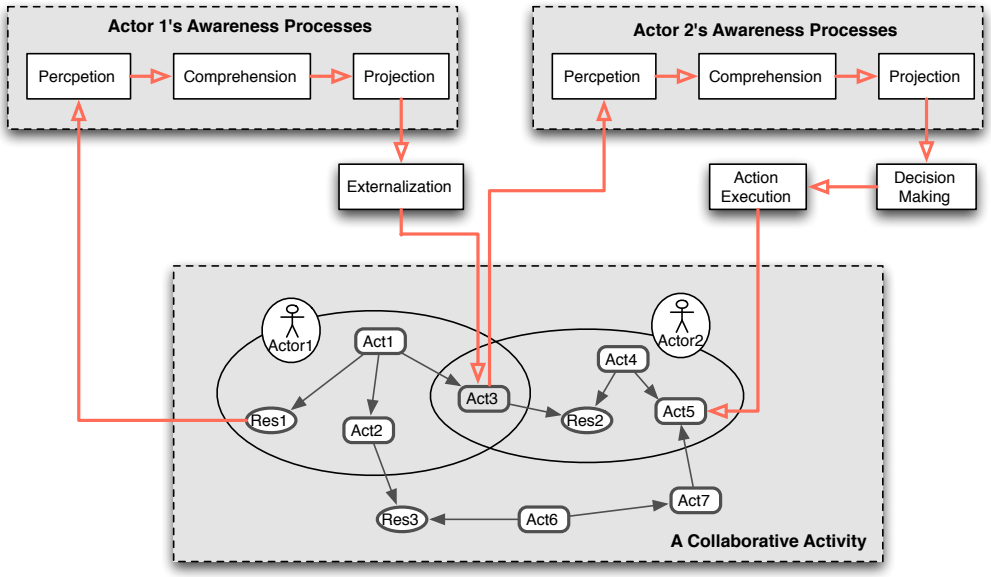


Figure 3.10. Awareness transaction via externalization

can be processed. Comparing with awareness transactions that merely rely on mutual monitoring, the externalization-based awareness transactions can support the exchange of intentional aspects of awareness, such as human actors' intentions to actions, or their interpretations of the situation. Comparing with direct communication, it has a lower level of obtrusiveness as it gives receivers control on when to consume the information.

3.3.2.2 Developmental trajectories

The development of awareness at the team level is seldom achieved by one single awareness transaction, rather most likely to entail a sequence of awareness transactions among multiple actors. In this way, we consider the development of collaborative awareness as developmental trajectories [104] that are triggered by some initial awareness information, and then the awareness knowledge evolves over time, contributed by a series of transactions.

Figure 3.11 illustrates the idea of a developmental trajectory including multiple awareness processes. In the beginning, *Actor*1 performs some action *Action*1 in the local scope, which is perceived by *Actor*2, as *Actor*2 is monitoring the status of *Action*1 (a transaction via *mutual monitoring*). Upon receiving the awareness element associated with *Action*1, *Actor*2 develops the individual awareness by interpreting how the awareness element on *Action*1 may impact his/her own work, and leads to some new awareness

element that he/she wants to pass to another actor *Actor*3. As a result, *Actor*2 initiates a direct communication with *Actor*3 (a transaction via *communication*), triggering *Actor*3's individual awareness process to understand the communicated message. The result of *Actor*3's individual awareness process is externalized as a new awareness element that is attached to a shared action with *Actor*4 (a transaction via *externalization*). As *Actor*4 is monitoring the shared action with *Actor*3, he/she perceives the awareness element passed from *Actor*3, starts the individual awareness process and leads to further changes on his/her action *Action*2.

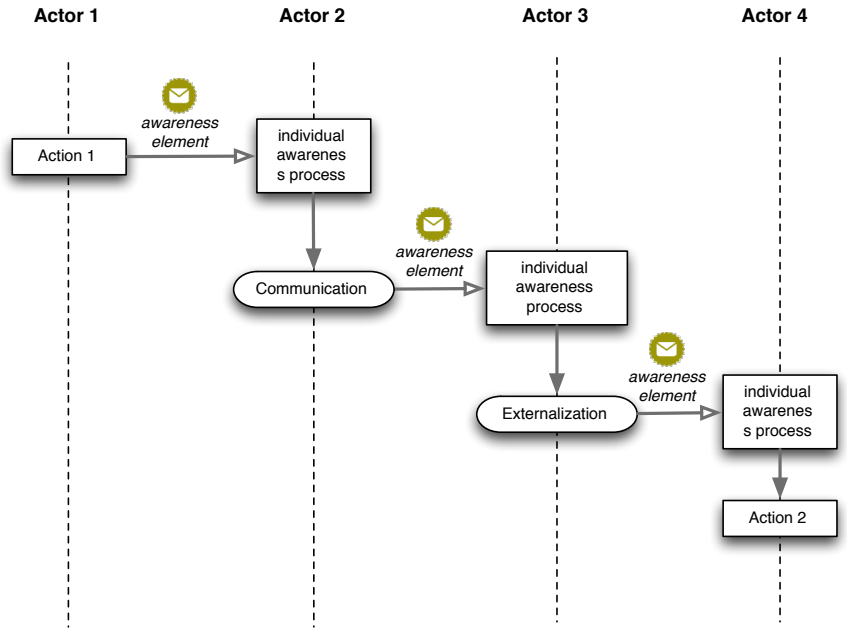


Figure 3.11. Developmental trajectory of awareness

Conceptualizing the development of awareness at the team level as developmental trajectories exhibit several important characteristics of the collaborative awareness phenomena:

1. The awareness knowledge at the team level is socially constructed through multiple awareness transactions. The awareness element perceived by *Actor*3 is different from the initial information received by *Actor*2, however they are related as the former is built on top of *Actor*2's interpretation of the latter. In this way, the awareness knowledge is undergoing continuous construction as the initial awareness information is enriched in every awareness transaction.

2. In one developmental trajectory, different types of awareness transactions can be

observed. In the example demonstrated by Figure 3.11, the transaction between *Actor*1 and *Actor*2 is based on mutual monitoring; then *Actor*2 directly communicates the awareness message to *Actor*3; last the transaction from *Actor*3 to *Actor*4 is via externalization.

3. The development of awareness is coupled with the development of the collaborative activity. In the example illustrated by Figure 3.11, the whole awareness development is triggered by the performance of some action, i.e. *Action*1. On the other hand, the development of awareness throughout the four actors then leads to *Actor*4 to perform another action, i.e. *Action*2.

3.4 Discussion

In this chapter, we present our conceptual model of awareness in large-scale distributed collaboration. Our conceptual model is based on the understanding of collaborative activities, from which we identify the major characteristics of the awareness phenomena and elaborate on the awareness processes through which the awareness is developed at both the individual level and the team level.

In Section 8.3, we apply the conceptual model in an emergency response scenario to understand the major characteristics and developmental trajectories of the awareness phenomena in large-scale distributed collaborative activities.

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 takes into account the awareness phenomena at both the individual level and team level. Each actor develops their individual awareness through individual cognitive processes and at the same time, the actors engage in team processes to conduct transactions to achieve collaborative awareness.

2. Second, the concept of local scope and dependencies allows us to define how the awareness is distributed across multiple actors. The different actors often attend to different sets of actions in their respective local scopes of work (*partiality*). At the same time, the actors' local scopes are often overlapped due to the shared actions, or various dependency relations may occur across multiple local scopes. This drives the occurrence of awareness transactions among multiple actors to achieve *compatibility*.

3. Last, Our model is built on top of the underlying model of collaborative activities. Our discussion of the major characteristics of awareness phenomena, and the various individual and team processes to develop awareness is centered around the concept of activity, local scopes of work, and dependencies.

Chapter 4

Awareness Promotion

The conceptual model of awareness presented in the previous chapter clearly shows the increased complexity of the awareness phenomena in large-scale distributed collaborative activities:

1. First, due to the highly distributed nature of the collaborative activities, team members play specialized roles and are engaged in different actions. As a result, each actor has distinct awareness requirement, related to the goals and actions that they are working towards.

2. Second, in order to achieve the compatible awareness at the team level, each actor needs to be aware of more things than their individual awareness, and spare extra effort to interact with each other.

3. Last, because of the continuous development of the collaborative activities, the actors' awareness requirements are quite fleeting.

As a result, awareness can no longer be achieved effortlessly by the actors themselves, and computer support can play an important role to augment and complement human capability. This chapter provides the overview of our computational approach to supporting awareness in large-scale distributed collaborative activities. 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 collaborative activities.

In this chapter, we first identify the major challenges for human actors to achieve and develop awareness in distributed, complex collaborative activities, which highlight the design aspects in which computer systems can provide support. Then we discuss how

these design aspects have been addressed (or partially addressed) in existing awareness supporting systems. Last, we present our computational awareness promotion framework that has the potential to better address these challenges.

4.1 Designing for awareness support

As described in Section 3.3, in order to achieve awareness, human actors have to engage in a variety of awareness processes at both the individual and team levels. In this section, we identify the major cognitive and interactive challenges for human actors in these awareness processes, and analyze how the computer system can provide support to address these challenges correspondingly.

4.1.1 Support for individual awareness

Perception The achievement of individual awareness starts with the ability of individuals to perceive key awareness elements in the environment. As the complexity of the collaborative activities scales up, there are potentially a very large number of awareness elements that are available at any given time. However, due to the partiality of individual awareness, each actor is only interested in a small set of them that are relevant to his/her own task. As a result, human actors have to focus on filtering out a large number of irrelevant information, which may consume too much of the actor's attention.

The computer can support the human actors in this process by reducing the number of awareness elements presented to the actors. If the system can predict the awareness needs of the actors, the effort of filtering out irrelevant information can be transferred to the computer systems, so that human actors can focus on processing only the relevant set of information to avoid information overload.

Comprehension Comprehension is to understand the meaning of perceived awareness information within the context of an actor's current actions [73]. In the comprehension process, new awareness information must be combined with existing knowledge about the current collaborative activity, which provides the inferential framework [21] for the actor to evaluate which part of the collaborative activity will be impacted by the perceived awareness information.

Without computer support, human actors have to maintain existing knowledge about the collaborative activities in their minds. However, in large-scale collaborative activities where the magnitude of the collaborative work can grow significantly as hundreds

or thousands of actors engaged in myriads of interdependent actions, it becomes a challenging task for human actors to maintain such an inferential framework in the mind. Alternatively, there are two ways that the comprehension can be supported by computer systems:

1. 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 [53], or pointers to locations of other important information in the external representation [62]. In this way, the limited working memory resource of human users is freed up for other aspects of cognition [62].

2. 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 [108]. Supporting comprehension by linking has its basis on the design principle of offloading cognitive processes onto perceptual processes [62]. 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 [53].

Projection Projection is the process that the individual predicts the future states of other actions based on the comprehension of awareness information. As argued by Endsley [33], this is the most difficult and taxing part to achieve individual awareness because it requires a fairly well developed mental model of the actions and relationships among them, along with the capabilities to perform reasoning. The analytical reasoning is central to the projection process, through which the actors explore possible alternative future scenarios and identify the one most likely to come [107]. Similar to the comprehension process, when the complexity of collaborative activities scales up, it becomes a challenging cognitive task for human actors due to the large volume of knowledge that needs to be stored and consumed during the reasoning process. The computer support for projection, as a result, can be supported from two aspects:

1. The projection process can also be supported by providing external representa-

tions of existing knowledge and the analytic tools to allow the actors to synthesize information and derive insight from it.

2. On the other hand, the system can perform the reasoning tasks on behalf of human actors, and simply present the reasoning results to them. In this way, the high level reasoning tasks are transformed into low level perceptual tasks for the actors.

4.1.2 Support for collaborative awareness

As we described in Section 3.3.2, the development of collaborative awareness is mediated by the transactions between multiple actors through which actors' individual awareness processes are connected as developmental trajectories. As a result, besides the individual awareness processes, the actors have to perform the team processes to conduct awareness transactions.

The challenges for actors to perform the team processes for awareness development can be analyzed from two aspects: the overhead of maintaining additional transactive knowledge and the extra effort to perform them.

Transactive knowledge In order to conduct awareness transactions, the actors need to attain more knowledge than what is needed for their individual work. In general, we can identify two types of knowledge that are necessary for the actors to conduct awareness transactions.

1. First is related to the concept of 'transactive memory' that describes how the individual memories of team members are supplemented by the additional knowledge about other actors [113]. Within the context of collaborative awareness, that means, in order to conduct awareness transactions, the actors need to maintain knowledge about how each other's work is related. To monitor each other's work, the actors need to know who else's work can potentially impact their own work. To externalize or communicate their individual awareness to others, the actors need to know whose work may be impacted.

2. The other is knowledge about the historical development of awareness. The awareness information perceived by an actor may not be directly generated from the environment, rather it can be the result of multiple awareness transactions across several actors. In order to understand the meaning of the perceived awareness information, the actor has to trace back to understand where it comes from, who else has been impacted or contributes to its development, etc.

As the complexity of collaborative activities scales up, maintaining these two types of knowledge increases the actors' cognitive overhead significantly. With the increased number of actors and possible dependencies among their actions, the knowledge about other actors whose actions are related to an actor grows rapidly. In addition, there are more actors participated in each developmental trajectory, so that it becomes difficult for the actors to keep track of the historical development of awareness.

Correspondingly, the computer system can support the actors by maintaining these two types of transactive knowledge for them. Whenever the actors need the knowledge, they can turn to the computer system to retrieve it. Moreover, the computer system can make use of the knowledge on behalf of human actors and perform reasoning to facilitate the awareness transactions. If the computer system knows whose actions to monitor, the system can monitor them for the actors. If the computer knows who needs to be notified by a piece of awareness information, it can deliver it to the relevant actors without human intervention.

Team processes Besides the additional knowledge that is needed, awareness transactions are often achieved by some sort of team processes, such as direct communication or externalization in which the actors make aspects of their individual awareness visible to others. These team processes pose an additional effort to the actors who initiate the awareness transactions. Moreover, the actors who perform these team processes, i.e. the *initiators* of team transactions, are often not the actors who will directly benefit from them. The disparity between cost and benefit [49] becomes an important challenge for the awareness development, as it may discourage the actors to conduct the awareness transactions. As a result, in order to support the development of collaborative awareness, the computer system needs to provide effective tools for the actors to perform the team processes, so that the extra effort can be minimized.

Table 4.1 summarizes the major challenges for human actors and the design aspects for computer systems to support the awareness processes.

4.2 Existing studies

By outlining the design aspects for awareness support in Section 4.1, we can use them to review existing awareness supporting systems by looking into how these design aspects have (or have not) been supported. Before that, we need to make a distinction between two types of computational models for organizing and processing awareness information:

Awareness Process	**Challenges for human actors**	**Design aspects for computer support**
Development of individual awareness	*Perception*: they have to filter out a large number of irrelevant information, consuming their attention resources	*1. Filtering*: the computer predicts the awareness needs and filters out irrelevant information for human actors.
	Comprehension: they have to maintain a large volume of knowledge about the collaborative activities in their minds	*1. Representation*: the computer provides external representations of the existing knowledge to free up human working memory resources. *2. Linking*: the computer presents the awareness information along with the contextual information that is potentially relevant to understand its meaning.
	Projection: they have to maintain the large volume of existing and derived knowledge during the reasoning process	*1. Representation*: the computer provides external representations and analytic tools to allow the human actors to synthesize information and derive insight. *2. Reasoning*: the computer performs the reasoning tasks and presents the reasoning results to human actors.
Development of collaborative awareness	*Transactive knowledge*: they need to maintain transactive knowledge that increases their cognitive overhead significantly	*1. Representation*: the computer provides external representations of the transactive knowledge. *2. Reasoning*: the computer makes use of the knowledge on behalf of human actors and performs reasoning to facilitate awareness transactions.
	Team processes: they need to spare additional effort to conduct team processes, but are often not the actors who will directly benefit from them.	*1. Tools*: the computer needs to provide the tools that simplify the effort to perform team processes.

Table 4.1. Design aspects for awareness support

space-based and *event-based* models, because it is the underlying computational model that determines how the awareness processes are supported in an awareness system [46].

4.2.1 Computational models of awareness support

Most awareness systems rely on certain computational models to organize and process awareness information. A computational model usually provides a representation of the collaborative 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.

4.2.1.1 Space-based models

Many researchers have previously described systems to support awareness based on the spatial metaphor [9, 84, 91, 98]. These space-based models explicitly represent the various shared objects, such as actors, information, resources, or other computer artifacts, situated and manipulable in some space. Awareness is then achieved by the interaction between actors, as they present themselves and attend to each other in the space [84]. 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 [26]. 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 actions of others, to perceive awareness information [98]. 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 [27]. 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 [84] 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 [13], and collaborative virtual environment [10]. 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 [13], 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 [50], 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 [26]. 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 [50].

4.2.1.2 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 [82]. 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 events, 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 shared spaces, 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 objects or artifacts in the collaborative activities, which provides several distinguishing features from the space-based models:

1. It allows the awareness information referring to any aspects of collaborative activities, not only the external aspects, but also the internal aspects reflecting the intentions and beliefs of actors.

2. The awareness information is not limited within the user's current viewpoint. 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 viewpoint.

3. The awareness information is pushed to the actors when the event happens, so that they do not need to spare their attentional resource to monitor each other's work.

Because of the decoupling of awareness information from the representation of collaborative work, many existing event-based systems actually do not need to provide an explicit shared representation of the collaborative work [78, 39]. Even in some systems where the collaborative work is represented (for example, the GroupDesk [43] and the POLIAwaC systems [100] use semantic networks comprised of actors, artifacts, events, and their relations; the BSBW system [12] and Atmosphere model [81] use hierarchical structures of workspaces), they are mainly used by the designers to specify events and manage event distributions, and are invisible to the actors.

Table 4.2 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 4.1 have (or have not) been supported each of these models.

	Space-based models	**Event-based models**
Representation of the collaborative work	explicitly represented as a 'shared space' inhabited by objects representing actors, information, resources, or other computer artifacts	implementation-dependent internal representations
	visible to users	invisible to users
Awareness information	embedded as perceivable properties of objects in the space	explicitly represented as discrete events
	limited to external aspects of shared objects	can refer to both external and internal aspects
	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 information is pushed to the users
Example Implementations	1. physical spaces (e.g. Portholes [27]) 2. virtual spaces (e.g. AROMA [75], DIVA [13], MASSIVE-2 [10]) 3. workspaces (e.g. ShrEdit [26], GroupKit [85])	GroupDesk [43], POLIAwaC [100], NESSIE [78], Elvin [39]

Table 4.2. The main distinguishing features of space-based and event-based models

One thing to note is that, in many existing awareness systems, the space-based and event-based models are not exclusive, rather they are combined together to provide an integrated awareness supporting environment. Many shared workspace systems, such as the GroupDesk [43], the POLIAwaC [100], and the ENI [46] systems, present a shared workspace among the collaborators, and at the same time support event notifications. Atmosphere model [81] uses hierarchical structures of workspaces, i.e. spheres, to repre-

sent the shared collaborative work, and they are also used to specify events and manage event distributions.

4.2.2 Awareness support in existing systems

4.2.2.1 Support for perception

The support for perception regards filtering awareness information for human actors, which has been described in both *space-based* and *event-based* systems.

Space-based systems Systems adopting the space-based awareness model usually depend on the various relationships between objects inhabiting in the shared space to decide on the relevance of awareness information.

Benford et al.'s spatial model [9] 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 awareness information is selected based on the overlapping between the receiver's *focus* and the performer's *nimbus* in certain medium [9]. 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 [84] 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 [98] 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*).

Event-based systems Filtering awareness information in event-based systems focuses on managing events subscriptions that are used by human actors to describe their awareness interests. Many mechanisms that enable the event subscriptions have been discussed in the literature, and they vary from each other depending on how the filters are defined.

1. *Type-based filtering.* In the GroupDesk system [43], Fuchs et al. define several filters that allow the limit of the flow of events. Definition of these filters 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 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 [49]. Furthermore, the type-based subscriptions tend to be too rigid or unable to adapt to the team development [2].

2. *Content-based filtering.* 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 filtering is the increased expressiveness to describe interests. Users can subscribe to event patterns, which can describe a set of events based on their attributes, rather than pre-defined event types.

3. *Context-based filtering.* ENI [46] 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.

Comparison The space-based systems rely on the users to monitor the shared spaces and perceive the relevant awareness information. 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. This provides more flexibility to handle increased level of dynamics. However, space-based systems become much more problematic when the level of complexity in collaborative activities increases as well. When the numbers of objects, actors and their actions are significantly increased, representing them in a shared space and relying on the users to monitor and filter awareness information from it becomes a challenging task.

Event-based model 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 attention to other's actions 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, event-based systems become less effective to handle high level of dynamics.

4.2.2.2 Support for comprehension

The computer support in the comprehension process comes from two aspects: providing external representations of existing knowledge to free up human working memory resources, and linking the awareness information along with the contextual information that is potentially relevant to understand its meaning.

Space-based systems Most existing space-based systems provide visual-spatial displays [53] 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 [53]. Second, they provide overview of the whole field of work, and therefore can be perceived as a whole and provide necessary background for comprehending awareness information [13]. 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 [62].

However, space-based systems also have limitations in supporting comprehension. First, we need to distinguish the difference between the context of *origin* of the awareness information and the context of *work* of the user [46]. 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 number of actors, artifacts, and activities increases significantly.

Event-based systems 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 external events should be collocated with process and planning representations [21]. 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.

Comparison 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. 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 [21].

4.2.2.3 Support for projection

To our knowledge, explicit support for projection is rarely discussed in existing awareness systems. The question about how new awareness information will lead to future changes in human actors' actions is usually considered as a cognitive task that is performed by human users. Since most of the space-based systems provide the external representation of the collaborative work, to some extent it can support projection by providing such an external information store. However, the information represented in these shared spaces is often structured around the objects in the space. Many types of knowledge that is useful to predict future changes, such as the dependency relationships between actions, are not represented.

4.2.2.4 Support for transactive knowledge

Transactive knowledge in space-based systems can be supported by providing lightweight awareness widgets [51], along with the shared spaces. For example, DIVA uses a virtual office table to display participants in a chat room, as well as visualize their contributions over time[13]. The Babble system [35] includes a 'social proxy' showing team member's level of contribution to a threaded discussion. These awareness widgets represent some aspects of the collaborators, so that the users can check them to figure out who they should interact with, or whether the actors are available for interruption. However, these lightweight awareness widgets only display specific aspects of social awareness, such as the collaborator's arrival, availability, involvement, but cannot adequately support action-oriented awareness aspects [21], such as the predicated status of their actions, the reasons behind their actions, or dependency relations between actions.

In event-based systems, transactive knowledge is usually represented as events that are related to the actors and their actions. For example, in AERE system [42], the status of other actors' actions is represented as activity events, so that the actors can maintain knowledge about the other participants through the events. However, the actors have to

store the information carried in these events in their mind, which is less efficient than the space-based systems. The benefit of event-based system, on the other hand, is the capable to store history information about the awareness development. In many event-based systems, such as GroupDesk [43] and NESSIE [78], an event history is available so that the users can analyze the event history storage to understand how the events are developed. However, most systems store events in the database as individual records, and do not provide links between the events. In order to understand how the events are socially developed through transactions, they have to link them together based on their own interpretations.

4.2.2.5 Support for team processes

As we described in Section 3.3.2.1, awareness transactions are supported by three types of team processes: mutual monitoring, direct communication, and externalization.

Mutual monitoring In space-based systems, the process of mutual monitoring, i.e. making consequences of individual activities apparent to other participants, can be achieved either through the direct video-audio links between actors [27], or through the artifact mediation [106].

In media space-based systems, monitoring can be supported by the networks of audio and video to provide rich representation of people and their immediate surroundings [27]. By seeing others through the media space, it is believed that people can get a sense of others' actions so that they can infer the consequences of their actions. 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 actions, it usually does not provide the visibility of the work artifacts involved in these activities. As a result, seeing each other through a media space does not necessarily means the consequences of their actions are visible.

As a result, a more common approach to mutual monitoring in space-based systems is through the common artifacts in the shared spaces [13]. In these systems, users' actions are organized around the common artifacts, which supports the mutual monitoring through feedthrough [25], i.e., when artifacts are manipulated in some actions, they give off information that informs others the consequences of these actions [106]. However, artifact-mediated feedthrough is limited to the aspects of users' actions that can be represented as some artifacts, the internal aspects of their actions cannot be supported.

In existing event-based systems, the process of mutual monitoring can be supported

by the system's capability to automatically capture events from the environment or the actors. For example, in the GroupDesk [43] system, each time the state of an object changes due to some action of a user, a new event is automatically 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. However, the aspects of actions are limited to the external aspects that can be captured by computer systems.

Communication 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 [50]. However, it is usually supported separately from the awareness systems as a standalone feature in collaborative applications.

Externalization The basic way to support externalization in space-based systems is annotations, which are defined as hypertext nodes that are linked to the objects in the shared spaces [121, 114]. Annotations allow the users to add their comments or interpretations on specific objects 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 externalize other aspects of individual awareness, such as the intentions to perform actions, or state changes of actions.

Rittenbruch et al. introduce and explore the notion of "intentionally enriched awareness" [83], which refers to the process of actively engaging users in the awareness process by enabling them to express intentions. "Intentionally enriched awareness" emphasizes the role of actors to make some of their internal states (intentions, reasons, etc.) along with their actions visible to others. Their AnyBiff prototypical system is one of the limited existing attempts to support externalization in event-based systems, which allows users to generate, share, and use a multitude of events 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 that much more aspects of individual awareness could be made visible by the users. Supporting externalization involves not only help users to express their intentions, but also make their interpretations of awareness information during comprehension/projection, or the results of their decision-making visible.

4.2.3 Summary

Table 4.3 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	Make the shared space explicitly visible to each user, and rely on human actors to pick up the relevant awareness information	Filter information based events subscriptions that are generated by human actors to describe their awareness interests
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, 21].
Projection	Visualization of shared spaces	rarely supported
Mutual monitoring	1. video-audio links [27] 2. artifacts mediated feedthrough [106]	System generated events about actions [43]
Communication	separately supported	separately supported
Externalization	annotations [121, 114]	intention-enriched awareness events [83]

Table 4.3. Existing studies in awareness support

From the review of existing systems, we can see that these two types of awareness models both have some limitations to support the awareness in large-scale distributed collaborative activities.

Supporting distributed awareness Space-based models rely on the users to monitor the shared spaces 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 actions of users are distributed and each of them is working on a different set of actions, as actively monitoring the shared space requires extra attention and efforts from the users. On the other hand, event-based models provides more control on the users to decide on what aspects of awareness information that should be presented, so that the users do not need to switch their attention to other's actions 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.

Supporting dynamics of awareness 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. On the other hand, event-based models largely depend on event subscriptions which usually need to be clearly defined in advance. When the level of dynamics increases in collaborative activities, the user's awareness needs are often in the flux of changes, as their activities evolve. Hence, the event-based models become less effective to handle high level of dynamics.

Supporting for higher-level individual awareness processes 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 where the scaling problem becomes significant, 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.

Supporting for awareness transactions 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 field of work so that the actors can mutually monitor each other. Less discussion has been given to the explicit representation of transactive knowledge, or the externalization process, which actually is an even more important aspect of awareness in many complex, real world collaborative activities [52].

4.3 The awareness promotion approach

Our approach to supporting awareness in large scale distributed collaborative activities aims to address these limitations in existing awareness supporting systems. We argue that the computer system needs to play a more active role to promote awareness among collaborators. In this section, we first present the rationale for awareness promotion, and then overview the computational awareness promotion framework.

4.3.1 From awareness support to awareness promotion

In existing awareness systems, the role of computer can be described in the metaphor of 'tool' that emphasizes the human achieves his/her goal through the computer application [15]. 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, i.e. 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 collaborative activities become more and more distributed and complicated, more is demanded of the human actors to control the computer. As we conceptualize the awareness phenomena as a distributed cognitive system (Chapter 3), each human actor usually only has partial knowledge about the whole collaborative situation, and it is unlikely that the actor knows what exactly should be done in order to achieve awareness at the team level. With only local knowledge, the actor 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 actions are changed. As a result, the user either uses the local knowledge to develop awareness that is often incomplete, incompatible, or even incorrect; or has to maintain more global knowledge beyond 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 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 awareness requirements in various awareness processes so as to provide adaptive 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 collaborative activity), and its own behaviors (i.e. the capabilities to reason about the user's awareness needs in the various awareness processes and adapt interaction and

information presentation techniques to support them).

In general, we use the term 'awareness promotion' to define the paradigm of awareness support where the computer plays an active role to mediate the awareness processes. While human actors still need to undergo the cognitive processes to develop individual awareness, and use it to make decision and perform actions in their own local scopes of work; the computer system takes the responsibility to maintain a collective picture of the whole collaborative activity, and utilizes this knowledge to mediate the various cognitive and social awareness processes among human actors.

Table 4.4 illustrates the major distinctions between the existing awareness support method and the awareness promotion approach.

Awareness processes	**Awareness support**	**Awareness promotion**
Perception	1. the *computer* shares information in a common space, and relies on the *human* to monitor the shared space to perceive information (*space-based* models) 2. the *human* subscribes to events based on interests, and the *computer* filters out events based on human subscription *(event-based* models)	the *computer* infer who the information is relevant to, and present only the relevant information
Comprehension	1. the *computer* provides the representation of the whole shared space 2. the *computer* links the awareness information to the context of its origin, and the *human* infers the connection between the context of origin and his/her work context	1. the *computer* only represents the aspects of the collaborative activity that is relevant to the human 2. the *computer* can infer the connection between the awareness information and the human's work context
Projection	the *human* performs the projection as their own cognitive process	1. the *computer* represents the aspects of collaborative activity that are useful for projection 2. the *computer* can performs routine inferences and presents the reasoning results to aid the human
Transactive knowledge	the *human has to* maintain their individual transactive knowledge in mind	the *computer* maintains the transactive knowledge at the team level.
Team processes	the *computer* provides the tools and relies on *the human* to decide on what to monitor (as receivers), or who should be communicated (as initiators).	the *computer* makes direct use of the transactive knowledge to decide on who should be notified by what awareness information.

Table 4.4. Awareness support vs. awareness promotion

From the table, we can clearly see the opportunities and benefits for active promotion at both the individual and team level.

Promoting individual awareness The promotion of individual awareness can happen in all the three individual awareness processes.

1. In the perception process, instead of relying on human actors to decide on what information is relevant to them, the system can predict the human actors' needs and present the relevant information to the actors. Comparing with space-based systems, this promotion strategy has the capability to filter out information for the actors to reduce the cognitive load. Comparing with event-based systems, this promotion strategy has two important benefits: (1) in these collaborative situations, the awareness information that is relevant to an actor may come from remote collaborators through multiple awareness transactions. The relevance of such awareness information is usually difficult for the actor to judge and describe as subscriptions in advance; (2) because of the dynamics of the collaborative activity, the interests of an actor in the awareness information may change frequently as the actor works on different actions. Predicting the actor's awareness needs as the activity proceeds reduces complexity to modify subscriptions.

2. In the comprehension process, instead of presenting a whole shared space to the actors, the system can tailor the representation to only include the aspects of the collaborative activity that is relevant to the human actor, so as to improve the human actor's interpretation even when the whole collaborative activity becomes complex. In addition, the system can promote the comprehension by inferring how the awareness information is related to the human actors' actions in the local scopes, so that the human actors do not need to perform the inference by themselves.

3. To promote the projection process, the system can also tailor the representation to only include the aspects of the collaborative activity that is relevant to the human actor's projection task. Furthermore, as the system represents the knowledge about the collaborative activity, the system can perform some of the routine inferences and present the results to the human actors, so as to convert some higher level cognitive tasks into some perceptual tasks for the actors.

Promoting awareness transactions The promotion of awareness transactions focuses on maintaining the transactive knowledge and the capability of the system to use the knowledge to distribute awareness information in the transactions.

1. The computer system can maintain the transactive knowledge at the team level, instead of for each actor to store the transactive knowledge in the mind. Whenever the actors need the knowledge, they can turn to the computer system to retrieve it. This becomes extremely important when the complexity of collaborative activities scales up, maintaining the transactive knowledge increases the actors' cognitive overhead significantly.

2. Moreover, the computer system can make use of the knowledge on behalf of human actors and perform reasoning to promote the awareness transactions. If the computer system knows whose actions to monitor, the system can monitor them for the actors. If the computer knows who needs to be notified by a piece of awareness information, it can deliver it to the relevant actors without human intervention. In this way, the human actors can only focus on externalizing the aspects of their individual awareness, and leave the effort of monitoring and disseminating information to the computer system.

4.3.2 The computational awareness promotion framework

To operationalize the awareness promotion approach, we propose a computational framework in this study. In order to actively promote awareness in the various individual and collaborative awareness processes as described above, it is essential for the computer system to maintain a formal computational representation of the collaborative activities. We base the awareness promotion framework on top of such a computational representation, and then develop different awareness promotion strategies to make use of this system knowledge in different awareness processes.

In general, the awareness promotion framework is built on top of two major components: (1) a computational representation of the collaborative activities based on the PlanGraph model [19, 20], and (2) 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 (Figure 4.1).

On one hand is how the computer constructs and develops the knowledge representation of the collaborative activities within the event-driven processes. In order to model the development of collaborative activities, the computational representation should sup-

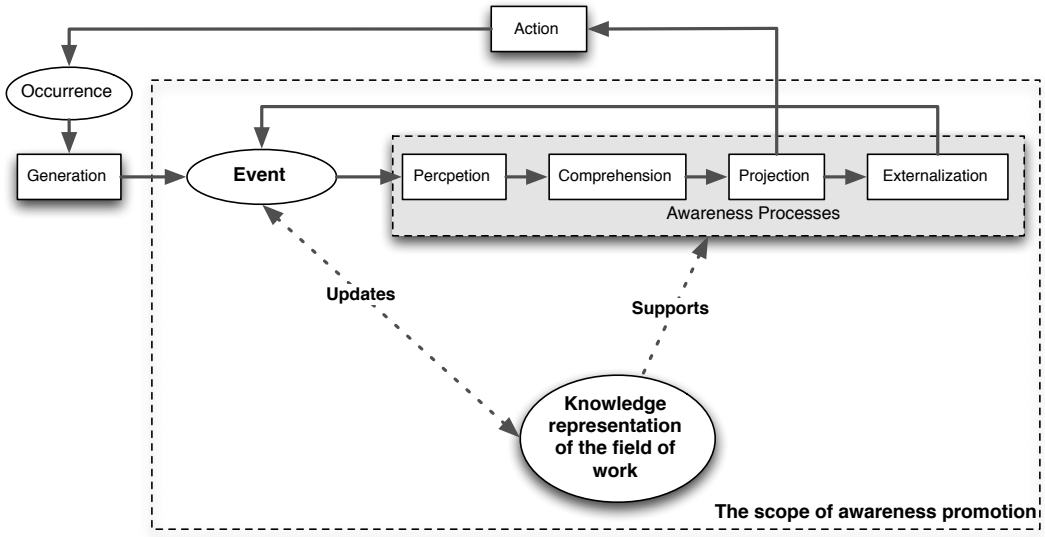


Figure 4.1. Awareness promotion framework

port dynamic adaptation so that it always reflects the current state of the changing environment. As human actors use a subset of the awareness information to develop their individual awareness, the system processes the awareness information that is available in the whole collaborative activity. The awareness information can be generated by sensing the occurrences in real world, or it can be the results of human actors' externalization of their individual awareness. All of the awareness information will be processed by the computer system to update its knowledge representation of the collaborative activity.

On the other hand, the knowledge representation is used by the computer system to promote the awareness processes. In general, the 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 collaborative activities, the system can offload some of the reasoning effort from the human actors. Meanwhile, the systematic knowledge can also be visualized to help the human actors to develop their individual awareness or externalize it for awareness transactions.

Following of this section discusses the major design choices of these two components, but leave their details to next two chapters respectively.

4.3.3 Computational knowledge representation

The nature of awareness phenomena in distributed, complex collaboration as we conceptualized in Section 3, and the goal of promoting awareness impose several requirements

for the knowledge representation of the collaborative activity:

1. The representation should encode enough knowledge about collaborative activities, including all the three constructs in our conceptual model, i.e. the basic elements and relations in collaborative activities, the local scopes of work for actors, and the various dependency relationships among actions.

2. The knowledge representation should be dynamically updated. Since the awareness needs of users change quickly with the development of collaborative activities, it is essential that the knowledge representation should be kept updating so that it always reflects the current state of the changing activities.

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

Existing awareness systems have provided several approaches to representing the collaborative work. The AREA system [42] models the collaborative work as semantic networks including relations 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 [81] describes the collaborative 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 [98] applies a reaction-diffusion metaphor to model the collaborative 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 actions and the dependencies using the semantic networks, but the local scopes of actors 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 actions. It supports the specification of each user's local scope using private 'spheres', but no dependency relationships between actions are supported. It supports the modification to the representation by human users,

but the system cannot automatically adapt the representation to the 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 collaborative activities that can satisfy all the three requirements. An important feature of our model is the capability to model intentions, beliefs, knowledge, and other attributes of actors' mental states [47]. As human actors' awareness processes are directed by their mental models, 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.

4.3.4 Event driven awareness processes

In our approach, we adopt the general event-driven interaction paradigm [36] to model the awareness processes. In the 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:

1. The event-driven paradigm opens up the opportunities for active promotion by the computational system. Instead of asking the human actors to monitor the environment and pull awareness information from it, events are pushed to them. The information push allows the system to take control and present the awareness information to the actors before they subscribe to them.

2. As awareness information is explicitly represented as first-class objects in event-driven approaches, this allows for representing any aspects of the collaborative activities, not only the external aspects, but also the internal aspects reflecting intentions and beliefs of the actors.

Our event-driven model of awareness processes shares some commonality with existing event-based models, as both use the concept of event 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 to distribute 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 also consumes the events. As to maintain the knowledge representation of the collaborative activities, the computer needs to take the events as input of its reasoning process, and use them to make inference and update its knowledge representation.

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 4.3.4.1.

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.

4. Existing event-based models treat events as discrete from each other. The system processes one individual event each time. After the event is disseminated to the corresponding actors, the processing on the event is done. However, as we conceptualize the awareness as undergoing continuous development, the events need to be considered as connected, and how they are built on top of each other needs to be tracked by the system.

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. In the following, we first clarify our concept of event, and then describe the event-driven awareness processes.

4.3.4.1 The concept of event

Before going further we should clarify what we mean by an event. The concept of '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 [101]. Each event in this notion is a message that describes a 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 [46] 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 actions, 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.

4.3.4.2 Awareness processes

Along with the distinction between the concepts of *'occurrence'*, *'awareness'*, and *'event'* is the notion of dynamic transformations between them (Figure 4.2). The transformations are tied to different processes in awareness development.

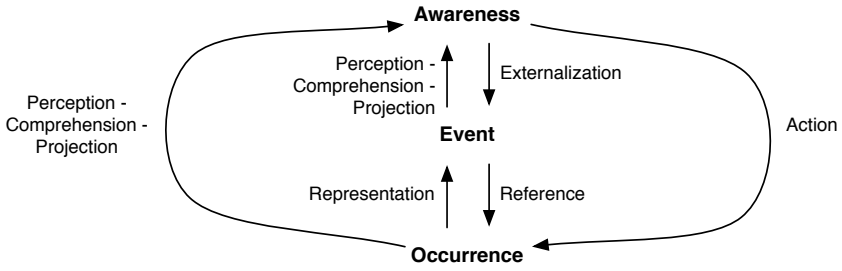


Figure 4.2. Transformations between occurrence, event, and awareness

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*. 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*. On 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 types of awareness transactions across multiple actors. The process of *mutual monitoring*

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 *externalization* 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 4.3 shows an example of how these event-driven awareness processes can be combined together to describe a developmental trajectory of awareness that involves three actors in a group. The awareness is propagated from *Actor*1 to *Actor*2 through the process of *externalization*, and is then propagated from *Actor*2 to *Actor*3 through the process of *mutual monitoring*.

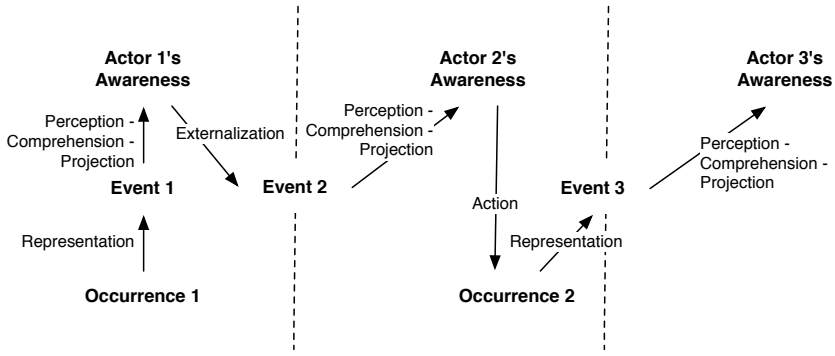


Figure 4.3. An example of event-driven developmental trajectory

4.4 Discussion

In this chapter, we first identify the major challenges for human actors to achieve and develop awareness in distributed, complex collaborative activities, which highlight the design aspects in which computer systems can provide support to augment and complement human capability at both the individual and team levels.

The analysis of the design aspects provides us the basis to review existing awareness support systems. The purpose of the review, however, is not to provide an exhaustive evaluation on existing studies. Instead, we use the review to understand how the design aspects have been addressed or partially addressed in existing awareness support systems, and identify the major limitations:

1. 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 large-scale distributed activities as we are interested in this study. On one hand, space-based models rely on the users to monitor the shared spaces and perceive the awareness information. This provides more flexibility to handle increased level of dynamics, but at the same time it becomes problematic when the collaborative is scaled up with higher level of complexity and actions of users are highly distributed. On the other hand, event-based systems are more efficient to handle higher-level of complexity as the users can decide on what aspects of awareness information that should be presented, and they do not need to switch their attentions to other's actions until the event notification happens. However, event-based systems become less effective in high level of dynamics as the user's awareness needs are often in the flux of changes.

2. The support for various awareness processes at both the individual and collaborative levels is limited. At the individual level, existing awareness systems have focused on supporting perception, the higher level of awareness processes are relatively less supported. 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' so that the actions of each other become visible to each other. However, less discussion has been given to support explicit representation of transactive knowledge, or the externalization process.

Motivated by these limitations, we argue that the computer system needs to play a more active role to promote awareness among collaborators, and provide an overview the awareness promotion approach. We believe the characteristics of awareness promotion approach provide the potential to address limitations of existing awareness systems to handle the scaled up complexity and dynamics, and to provide integrated awareness support. First, the awareness promotion approach utilizes the computational knowledge representation to model the collaborative activities and offloads some of the representation and reasoning efforts from the human to the computer. Hence, it can handle more complex collaborative configurations than existing awareness models. Meanwhile, the knowledge representation is dynamically updated to reflect the current state of the collaborative work, which allows it to handle increased level of dynamics. Last, as the computer's knowledge representation is at the team level and is equipped with the computational reasoning capabilities, it allows the computer to provide support on both the

higher level of individual awareness processes and awareness transactions.

The next two chapters provide details of this awareness promotion framework. Chapter 5 describes the knowledge components, i.e. the computational representation of the collaborative activities and the specification of events, and the mechanisms for updating the knowledge representation with events. Chapter 6 describes the different promotion strategies making use of the knowledge representation.

Chapter 5

Knowledge Representation and Updating

In this chapter, we provide details about the first component of our computational framework for event-driven awareness promotion, i.e. how the computer constructs and develops the knowledge representation of collaborative activities within the event-driven processes. We first provide a computational representation of collaborative activities using the PlanGraph model. Then, we discuss the specification and typology of events in our approach. Based on these two knowledge components, we discuss the knowledge updating process, in which they are mutually developed. On one hand, the knowledge representation of a collaborative activity is updated by reacting to the various external and internal events, so that it always reflects the current state of the collaborative activity. On the other hand, during the updating process, the system's knowledge about the events is also enriched by interpreting their meanings in the context of the collaborative activity.

5.1 Representing collaborative activities

In this section, we present the knowledge representation of collaborative activities based on PlanGraph model. By modeling an collaborative activity within a PlanGraph, the various entities and relations as we formalized in Section 3.1.1 can be represented as different components of the PlanGraph. Then we show how the local scopes and dependency networks can be derived from the PlanGraph model, making use of the knowledge about these entities and relations.

5.1.1 The PlanGraph model

The PlanGraph model with its basis in the SharedPlans theory is designed to represent the dynamic knowledge in the human-computer collaboration [20]. In general, a PlanGraph represents the knowledge about a collaborative activity towards a shared goal in a hierarchical way (Figure 5.1). The root of a PlanGraph is the overall goal of the actors in a collaborative activity, which is decomposed recursively into actions through the adoption of recipes. The PlanGraph as a whole represents the shared plan corresponding to the top-level collaborative activity, while each sub-tree with an action as the root represents the shared plan for that action. A PlanGraph is composed of three types of nodes: (1) *action* nodes represent all the actions and sub-actions in the collaborative activity; (2) *parameter* nodes represent informational or physical objects that are used by actions; (3) *condition* nodes represent states 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 sub-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 correspond to all the constraints that need to be satisfied before the action or any sub-actions can be performed.

2. Each parameter can be decomposed into sub-actions and conditions. The sub-actions of a parameter are used to assign values for the parameter, i.e. they are used to satisfy the knowledge-precondition on the parameter. The conditions attached to a parameter are other preconditions 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 ensure the condition is satisfied.

The PlanGraph model also encodes the actors that are participated in each 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. Each action node in a

PlanGraph includes several attributes to store the relations with participating actors as defined in Section 3.1.1.2: *Intentions* record the different intention relations of each actor towards the action. *Capabilities* indicate the level of capability of each actor to perform the action.

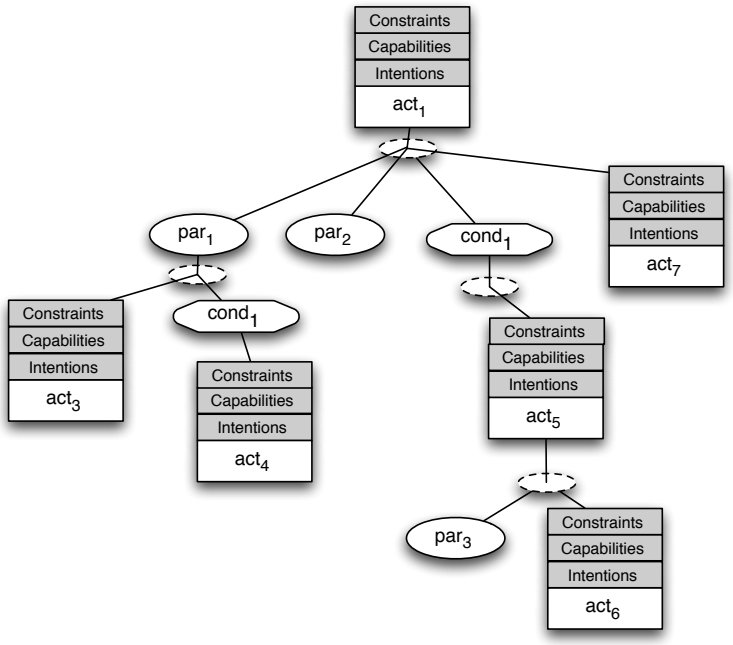


Figure 5.1. Structure of a PlanGraph

The PlanGraph model has been used to model the human-computer collaboration as a collaborative activity in several applications, e.g. the natural conversational interface to geospatial databases [20], collaborative dialog-based system to communicate vague spatial concepts [19], and context-aware mobile mapping [116]. However, in order to model the collaborative activities among human actors, several modifications to the original PlanGraph model have been made in this study:

1. First, the parameter nodes in the original PlanGraph model are mainly used to represent knowledge preconditions, i.e. they represent the knowledge needed for the action to be performed. In this study, we extend the concept of parameter to represent both the physical and informational resources that are needed for the action.

2. We extend the original PlanGraph model to explicitly represent conditions as standalone nodes. This allows us to model the indirect dependency relations among

actions between human actors. For instance, even though act_1 does not indicate a higher level goal of act_2, it can still depends on the act_2 because act_2 satisfies a condition that is required for performing act_1.

3. The original PlanGraph model for human-computer collaboration treats the computer system as an actor with its own intentions and beliefs. However, as we focus on tracking the states and relations among human actors' actions, we do not explicitly model the computer's beliefs and intentions.

5.1.2 Representing elements and relations

The PlanGraph model allows us to represent the basic entities and relations in a collaborative activity. The actions, actors, and resources are first-class objects in the PlanGraph model, and the multiple relations among then can be represented as structural relations in the PlanGraph model (Table 5.1).

Actors Each actor is represented as 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 directly modeled as a type of nodes in the hierarchical structure of the PlanGraph. Each action node has several important attributes:

1. The *goal* of the action is represented as an expression, indicating the expected effect of the action when it is successfully performed.

2. The *execution state* of the action records the current state of the action towards its performance, e.g. whether it has been started, successfully performed, or in the progress.

3. The *recipe* points to the current recipe to perform the action if the current recipe is selected from the knowledge base. Each recipe specifies a particular way to perform the action. In the case the recipe is identified by human actors on the fly, this could be empty.

4. The *Intentions* and *Capabilities* record the different relations between participating actor towards the action.

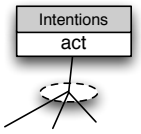
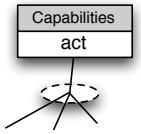
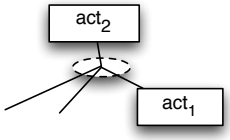
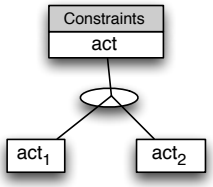
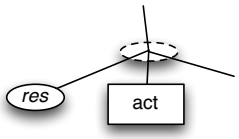
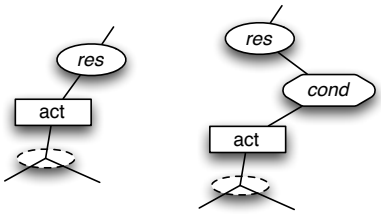
Relation	Structure in PlanGraph
$Pot.Int(ar, act)$ $Int.Th(ar, act)$ $Int.to(ar, act)$ $Perform(ar, act)$	search the *Intentions* slot attached to each action node: 
$Knows(ar, act)$ $Able(ar, act)$ $Workable(ar, act)$	search the *Capability* slot attached to each action node: 
$Sub.Act(act_1, act_2)$	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. The *Constraints* record the relations between sub-actions, such as temporal orders between them, or whether they can be optional.

Resources Resources are assigned to parameter nodes as values. Parameter nodes can be collective or individual. A collective parameter includes a collection of resources with the same type as its value, for instance, a parameter representing all the victims in the same rescue operation. All the resources assigned to the same parameter node need to satisfy conditions that are linked to the resource node.

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 actors 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 list of its sub-actions under current plan. The precedence relation that reflects the temporal order of performing two actions is encoded in the *constraints* attribute of their parent action node.

Relations between resources and actions The relations between resources and actions can be inferred from the structural relations between action nodes and parameter nodes in a PlanGraph. The resources assigned to a parameter node underlying an action node can be consumed by the action and all its sub-actions. The actions that are used to assign values to a parameter or achieve preconditions of the parameter can manipulate all the resources attached to the parameter node.

5.1.3 Constructing local scopes

Based on the definition of local scopes in Section 3.1.2, the local scope of an actor can be dynamically constructed by depth-first traversal through the PlanGraph structure to search for all the actions that are related to the actor through intention or capability relations.

We define a procedure $BUILD\text{-}LS$ 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 $ADD\text{-}To\text{-}LS$ that takes an actor object

ar and an action node *act* in the PlanGraph to add the action to the local scope *LSI* or *LSC* based on whether the corresponding intention or capability relation exists. The *ADD-TO-LS* procedure then calls itself recursively 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* starts with the initialization of *LSI* and *LSC*, and then recursively calls *ADD-TO-LS* from 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
```

5.1.4 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 $DN = (V(DN), E(DN))$ 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. $E(DN) \rightarrow V(DN) \times V(DN)$. 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 also be achieved by depth-first traversal 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 duplicates, the $BUILD\text{-}DN$ procedure focuses on building $E(DN)$. 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 *act*, i.e. to find all the dependencies in which the *act* is 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
```

5.2 Representing events

In Section 4.3.4.1, we 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. In this section, we focus on how events are computationally represented in this study. Although they share the same representational structure in general, different types of events are represented differently. In the following, we first describe the general representational structure of events, then discuss the major event types, and their differences in representation.

5.2.1 Structure of events

We follow many existing event processing systems to represent each event as a structured object consisting of a named set of attributes [68]. 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: 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 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 5.2.2.

While arbitrary attributes can be included in each event type, we can group them into three categories, following the definitions in [36] (Figure 5.2):

1. The *header* consists of generic information about the event, such as the event type, occurrence time, etc. The header attributes are not specific to a particular event type, rather shared by all event types.

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 the awareness system can use to enrich an event object with extra contextual information, such as human-readable explanation, multi-media content etc.

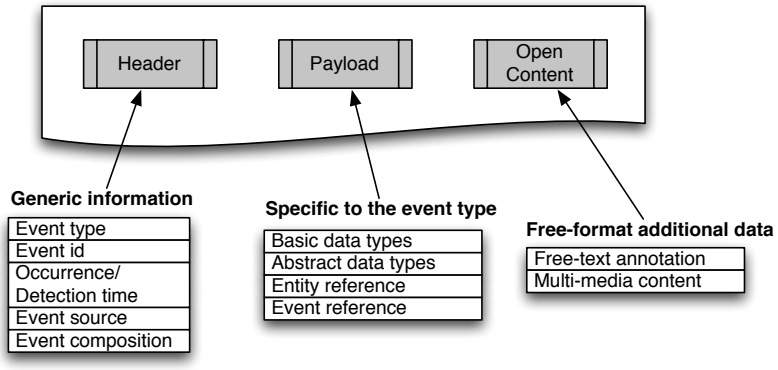


Figure 5.2. The structure of an event (adapted from [36] p.63)

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.

3. *Occurrence/detection time*. The occurrence time is the time when 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, recording 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 other 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.

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 collaborative activities. For example, an *'ActionPerformed'* event may use the reference to the action node stored in the PlanGraph model to indicate which action has been performed.

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

Open content The open content of an event can include any attributes an 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 human-readable explanation of their interpretations.

5.2.2 Event types

As we argue in Section 4.3.4.1 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 sets of attributes; (2) they are consumed differently by the system when updating the knowledge representation; (3) and they are treated differently in human users' awareness processes as well.

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 events by providing them a higher-level abstraction [68]. 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, both external and internal, and then discuss the composite events as a special event type with its own payload structure.

5.2.2.1 External events

As we conceptualize a collaborative environment as consisting of a variety of entities and relations, external events can be defined to indicate any kinds of changes on either these entities or 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 state 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 5.3 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.

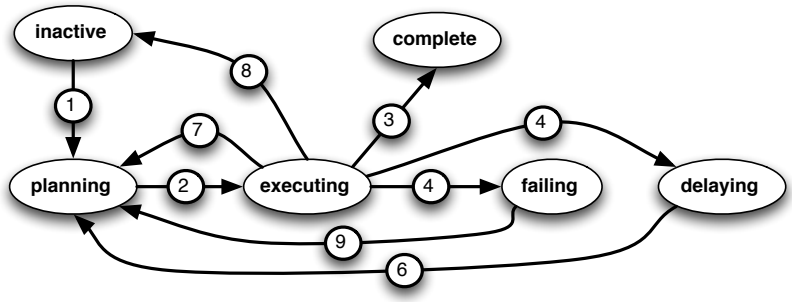


Figure 5.3. Execution state transition of an action

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 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 becomes holding, 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 [108].

1. The spatial relations link the entities through their spatial positions. The basic types of spatial relations have been well studied in the literature of geographic information systems, including binary topological [28], directional [40], and distance relations [55]. Topological relations are 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 covers 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 5.4 shows an upper level typology of the external events that can be defined on entities and relations in a collaborative environment.

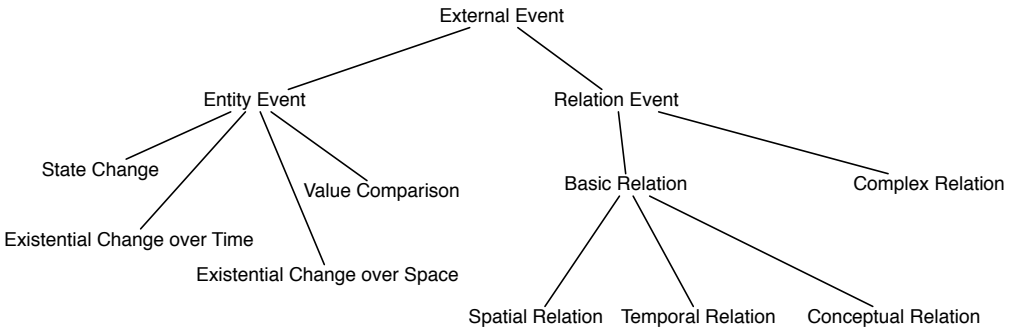


Figure 5.4. An upper level typology of external events

Relevance to the collaborative activities 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. As a result, the goal of identifying external events in an awareness system focuses on finding a subset of event types that could possibly contribute to the understanding of supported collaborative activities. In general, we believe that

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

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

2. *Direct change to relations in a collaborative activity.* Some external events are directly related to the various relations between resource, actors, and actions as we described in Section 3.1.1.2. 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 type can be used to describe relations between an action and a resource, i.e. the resource has been assigned to the performance of the action.

3. *Goal activation.* Aside from the two cases that external events can be directly linked to the entities and relations in a collaborative activity, external events can also impact the collaborative activity by activate the goals to perform actions. For example, an external event indicating that the fire alarm is ringing will activate the human actor's goal to escape from the office. In this case, the fire alarm is not directly linked to any entities in the human actor's activities, rather it motivates the actor to perform a new action.

4. *Implied changes to entities and relations.* In some cases, external events can also provide some evidence implying changes in the entities and relations in a collaborative activity. 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 in trouble.

5.2.2.2 Internal events

Internal events are used to describe the results of human actors' awareness processes. Unlike external events that can describe changes in any entities and relations in a collaborative activity, internal events describe the changes of human actors' internal mental states towards the collaborative activity. Internal events are usually derived from external events to indicate human actors' interpretations on them. 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 a collaborative activity. Figure 5.5 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 $Pot.Int$, $Int.Th$, or $Int.To$, a reference to the actor who adopts this intention, and a reference to the intended action. An optional free-text attribute is included in the open content part, where the human actor can provide the rationale for adopting the corresponding intention.

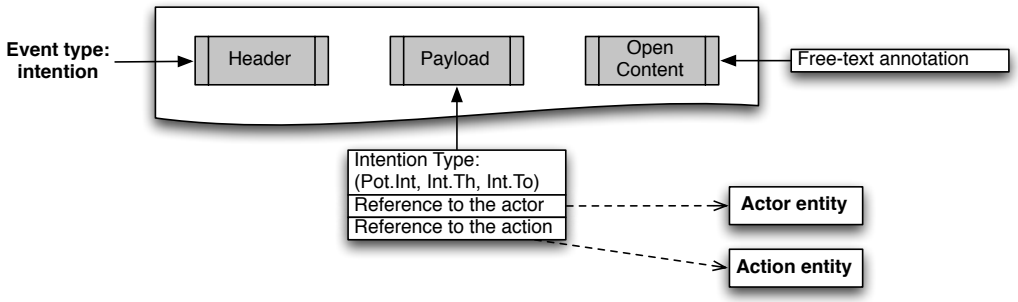


Figure 5.5. Structure of an intention event

A *belief* event describes a human actor's belief in some entities or relations in a collaborative activity. For example, it can be used to indicate an actor's belief that an action has been successfully performed, or the belief that the actor has the capability to perform an action. As belief events usually refer to some changes on entities or relations in a collaborative activity, we represent each belief event by embedding an external event representing the content of the belief in its payload (Figure 5.6). However, unlike the standalone external event that indicates a change that has already happened, the change described in a belief event can be something that will happen in the future. These belief events can be used to represent the results of the human actor's projection process, i.e. what the actor 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 the actor is about this belief.

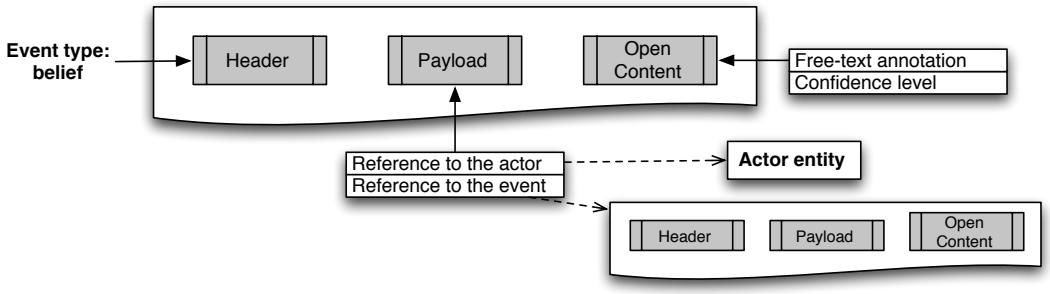


Figure 5.6. Structure of a belief event

5.2.2.3 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 simple 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 occurs. A more complicated composite event language can be found in [68] that includes different relations between 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 5.7).

5.3 The knowledge updating process

The knowledge updating process describes how the aforementioned knowledge representations of collaborative activities and events are mutually developed. 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 current collaborative activity 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.

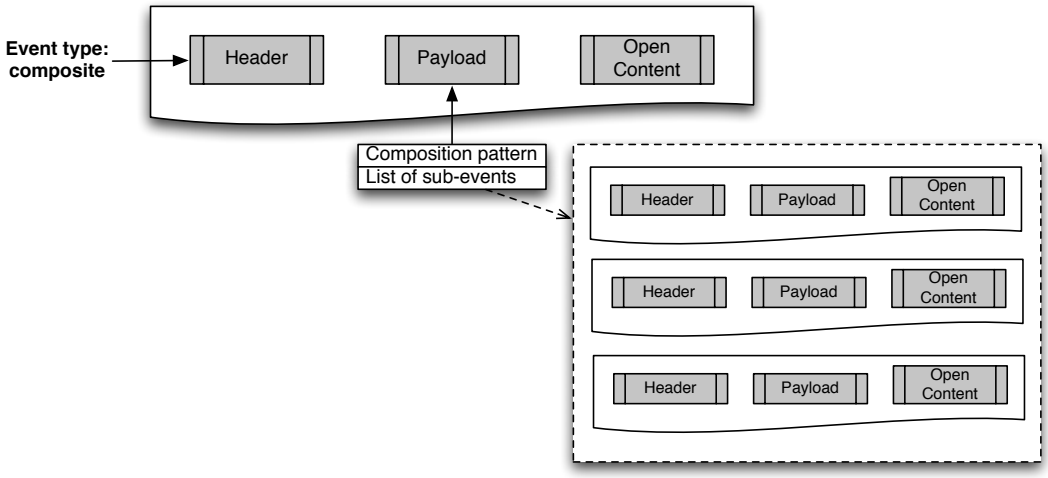


Figure 5.7. Structure of a composite event

Updating knowledge about the collaborative activity In general, the knowledge updating is a four-step process: *association*, *assessment*, *elaboration*, and *propagation*, through which the knowledge representation of the collaborative activity, i.e. the PlanGraph model is updated (Figure 5.8).

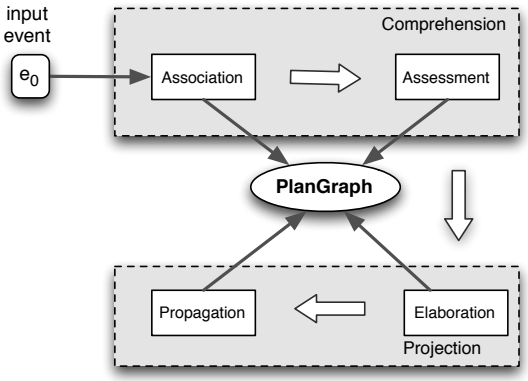


Figure 5.8. The knowledge updating process

1. *Association.* The knowledge updating starts with the association of an event with the PlanGraph model. In this step, the system searches the PlanGraph for an appropriate match between the input event and the entities and relations in the collaborative activity. 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 lead to new changes in the collaborative activity. For instance, it may trigger new actions that need to be performed, or change the current states of existing actions.

3. *Elaboration.* Based on the assessment of the event, the elaboration step is to reason about the system's expectation on how the newly added action should be decomposed into sub-actions, or what actors will be potentially involved in these new actions. This is usually performed with domain specific knowledge, such as recipes of action performance, and role specifications of actors.

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

The four steps of knowledge updating process share some commonality with the human actor's awareness development processes. The *association* and *assessment* steps are similar to the *comprehension* process, where human actors comprehend or understand the relevance of awareness information in relation to their tasks and goals. The *elaboration* and *propagation* can be considered as the projection of states in the near future. 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 actors usually only have partial knowledge about the collaborative activity, the computer system aims to possess the knowledge of the whole collaborative activity through knowledge updating.

One thing to note is that not every event will be processed in all the four steps. Some external event may not directly link to any entities in the collaborative activity, or the system does not have the complete knowledge to assess its implications in the collaborative activity. In such case, the event may be passed directly to human actors for interpretation. The result of human actors' interpretation may generate a new internal event that starts a new round of knowledge updating, in which the system's knowledge is updated.

Updating knowledge about the event As we emphasize in the beginning, while the knowledge about the collaborative activity is updated by the new event, the event itself is also developed during the system's reasoning process. For example, in the *assessment* step, an external event *'TrafficBlocked'* causes the system to believe that the action to deliver a resource cannot be achieved. On one hand, this causes the system to modify

the state of the delivery action in the PlanGraph. Meanwhile, the system generates a new internal event describing the system's belief about the state change of the delivery action. Later, in the *propagation* step, the system may generate another internal event to indicate that because of the state change of the delivery action, the action that depends on the resource delivery will also be impacted. In this way, the original event is derived into a chain of events as the knowledge updating 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 beliefs on how the other entities or relations in the collaborative activity have been changed due to the original event. In the propagation step, the system predicates the future state changes, and attaches more *anticipatory events* to the event chain. Figure 5.9 shows how an event is developed into an event chain in the knowledge updating process.

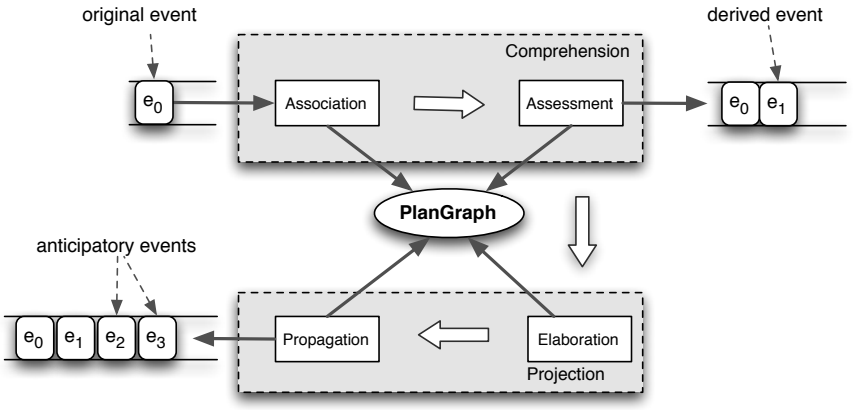


Figure 5.9. The development of an event chain

In order to record an *event chain*, we add three additional attributes in the *open content* section of every event in the chain:

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 the *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 at the end of the event chain.

Figure 5.10 shows how the additional information about the event chain is included in an event's representation.

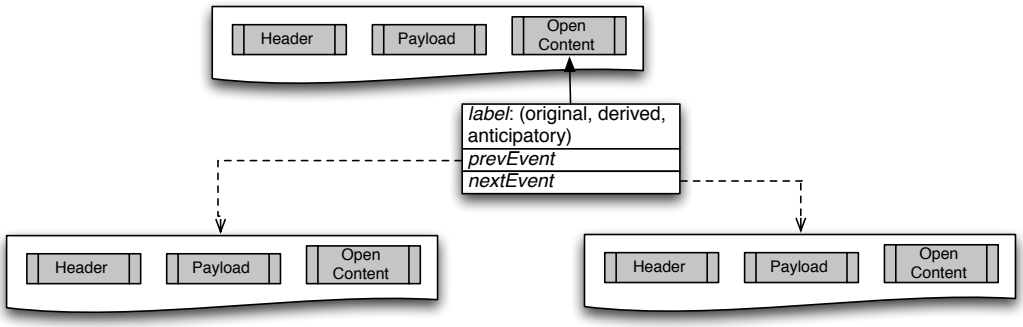


Figure 5.10. Event structure in an event chain

5.3.1 Association

The knowledge updating starts with establishing associations between the input event and entities and relations in the current PlanGraph model representing the collaborative activity, and uses the information carried by the event to update the PlanGraph model. Based on the different event types, the association is conducted differently.

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

1. If the event indicates an existential change over time, the event is directly passed through to the assessment step, as the entity related 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 the 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 the event is direct linked to the matched PlanGraph node.

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 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 is related to a structural relation, e.g. an action act_2 is a sub-action of another action act_1. We search all the action nodes in the PlanGraph to find any match with act_1. 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 a sub-action to achieve a condition, or a new parameter 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* attribute associated with the action node to find whether any existing actor has the same identifier as the actor 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 *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 whether 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 capability level into 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, if an association between the event and the PlanGraph model is found, two tasks are performed. First, the corresponding PlanGraph entity or relation is updated based on the new information carried by the event. Second, the event is enriched by directly linking to the corresponding PlanGraph node, as the result of which the context of origin for this event is identified. However, not all the events can be directly associated with the PlanGraph model. Some external events may describe changes on the entities that are not currently in the PlanGraph, but implicitly impact the collaborative activity by motivating new actions or implying changes. These events will be passed to the assessment step for further analysis.

5.3.2 Assessment

In the assessment step, each event is evaluated to check how it can lead to new changes in the collaborative activity implicitly, where inference becomes necessary. For example, an external event indicating that a resource is now located at an actor's position implicitly indicates the successful performance of the resource delivery action to the actor. 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 in different application domains:

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, the system derives a new state change event on this action, and adds 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, it derives a new state change event on this condition, and pushes it into the event chain.

3. *Parameter nodes.* If an event describes changes on a 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, it derives a new state change event on this parameter, and adds 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 actions that need to be added to the

collaborative activity. This type of events is usually called *triggering events*, as they are often not directly associated with any existing nodes in the PlanGraph, but will trigger some new action to be added. 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 is searched through the activation rules to find whether it satisfies any of the conditions. If so, a new action is added to the PlanGraph, and a 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 [11], pattern recognition [120], or case-based reasoning [59], to enhance the system's reasoning capabilities. However, these reasoning techniques often require a large amount of domain knowledge to be modeled, and lack flexibility to handle unexpected events. As a result, in the assessment step, we design the system to focus on more reliable low-level routine inferences, and leave the complex, higher level assessment tasks 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 sent to human actors for interpretation. The result of human interpretation generates new events that are then sent back to the system to update the system's knowledge.

5.3.3 Elaboration

The main goal in the elaboration step is to advance the collaborative activity from the system's side. Based on the specification of SharedPlan theory [48], 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 predicting the 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 sub-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 collaborative activity because of some triggering event in 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 elaborate the collaborative activity by adding new action, parameter, or condition nodes into the PlanGraph model; and the knowledge about role specifications 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 notify these actors about the events.

The elaboration process is predictive as it reflects the system's prediction on how the collaborative activity will be advanced due to the occurrence of the event. The system provides a default plan of performing an action, or 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 modify their intention or capability towards the action in the PlanGraph.

5.3.4 Propagation

Propagation is another predictive process that the system can perform to predict future state changes in the collaborative activity. It is triggered by the events that either directly indicate (in the association step) or imply (in the assessment step) state changes on the actions, 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: 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 also be changed to *failed* if the plan is not changed.

The propagation is performed on the dependency network, which is constructed from the PlanGraph model (Section 5.1.4). The dependency network abstracts away the detailed information on each action node and focuses on the dependency relations among them. As a result, 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 collaborative activity, 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 system's belief in the states for every other node after some state change occurs on a node.

The operationalization of the Bayesian network includes three tasks: (1) the construction of the Bayesian network, (2) assignment of conditional probabilities for each link, (3) and the performance of belief updating when some events arrive.

Construction of Bayesian networks By following the algorithm in Section 5.1.4, we can construct the dependency network from the PlanGraph model, and then the con-

struction of Bayesian network is very straightforward. We translate each basic element in the dependency network into a node variable, and the different dependency relations 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 5.11 shows all the states for each type of node and the possible state transitions.

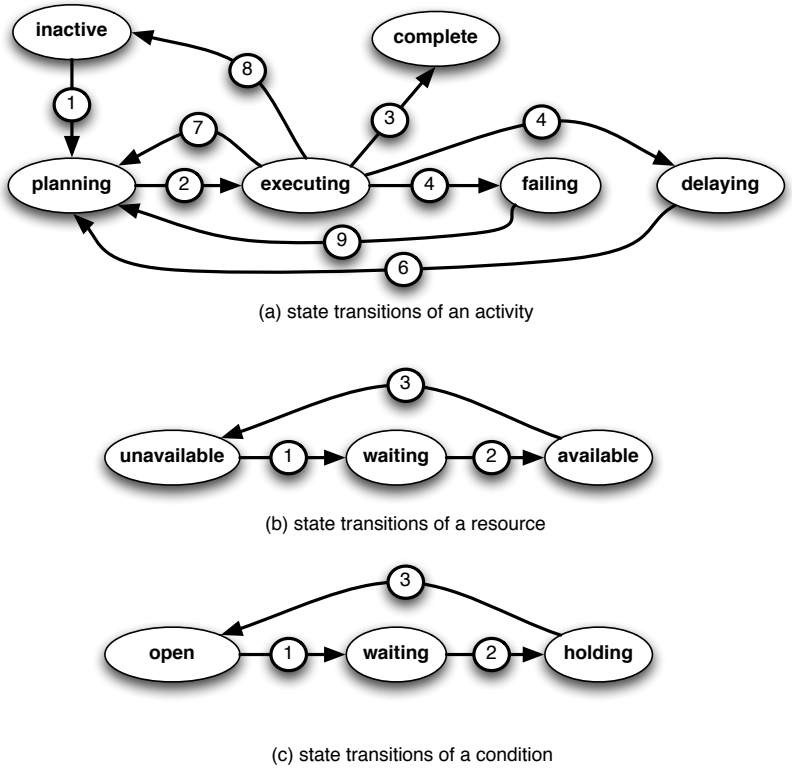


Figure 5.11. Types of node variables and possible values in a Bayesian network

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.

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 state changes that can be later overwritten by human actors' interpretation. When a human actor changes the belief in the execution state of an action, the change will overwrite the system's 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 Pearl's 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 depending on the action associated with the initial state change. The evidential propagation occurs whenever a human actor modifies the system's prediction on a given

node and the system uses it as an evidence to trace back and revise the previous beliefs on other actions.

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 uses 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 5.12 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.

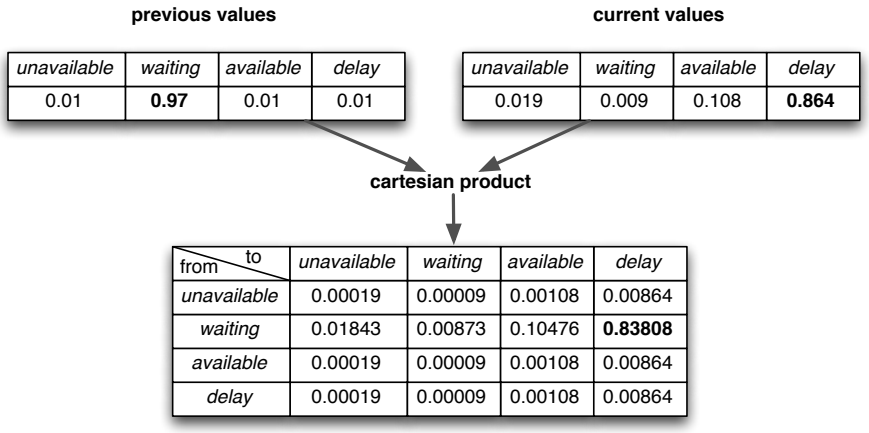


Figure 5.12. An example of calculating state change probabilities

5.3.5 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 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 with one medical professional (*med*) working at

it. There might be several drivers ($dr_1, dr_2, ...$) 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 5.13 shows the knowledge updating process performed on this event.

1. When the event is first sent to the system, the system attempts to associate it with any existing entities in the PlanGraph. 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 on existing actions or trigger new action. By checking the association rules stored in the knowledge base, the system finds that every *'NewVictim'* event will activate the goal to rescue the victim. 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 PlanGraph, which triggers the elaboration process. The system searches the knowledge base to find a recipe for the *'rescue'* action, and extends 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 parameters 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 5.13, 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.

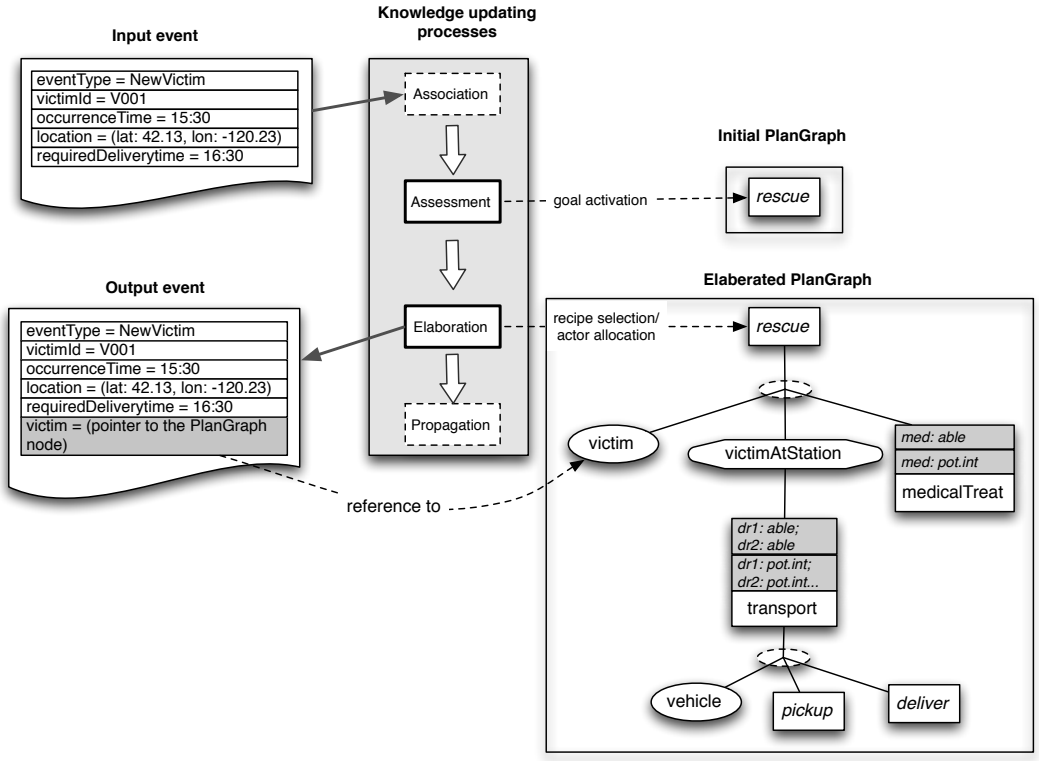


Figure 5.13. The knowledge updating example (*Event 1*)

Event 2: *'IntentionChange'* The second event is an internal event that is generated by the driver dr_1. After interpreting the first *'NewVictim'* event, dr_1 intends 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 dr_1 intends to perform (*'transport'*), and the intention type (*int.to*).

As the system receives this event, it first attempts to associate the event 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 action (*'transport'*) mentioned in the event and the action nodes in the PlanGraph. As 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 reference to the corresponding 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 5.14 shows the whole process performed on the second event.

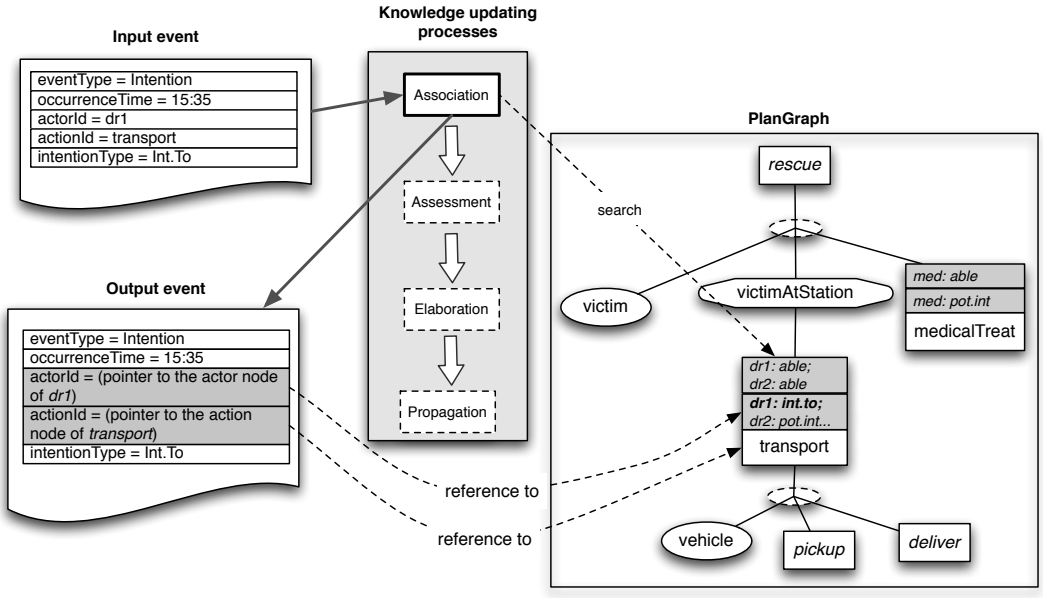


Figure 5.14. The knowledge updating example (*Event 2*)

Event 3: *'FuelLevelLow'* The third event happens after 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 5.15 shows the knowledge updating process performed on this event.

1. When the event is first sent to the system, the system attempts to associate it 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 the 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 *'ExecStateChange'* is derived to describe the execution state of the parameter, and is 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 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 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 5.15, after the knowledge updating process, the initial event is augmented into an event chain with three events: the original external event *'FuelLevelLow'*, the state change event on the parameter *'vehicle'* derived in the assessment step, and the anticipatory event predicting the state change on the action *'pickup'* in the propagation step. This example shows the idea that not only the knowledge representation of the collaborative activity, i.e. PlanGraph model is modified during the knowledge updating process, but also the original event is enriched with system generated knowledge.

5.4 Discussion

In this chapter, we first focus on the knowledge representation of collaborative activities. We employ the PlanGraph model to represent the entities and relations in collaborative activities, and then use this model to derive knowledge about local scopes of work

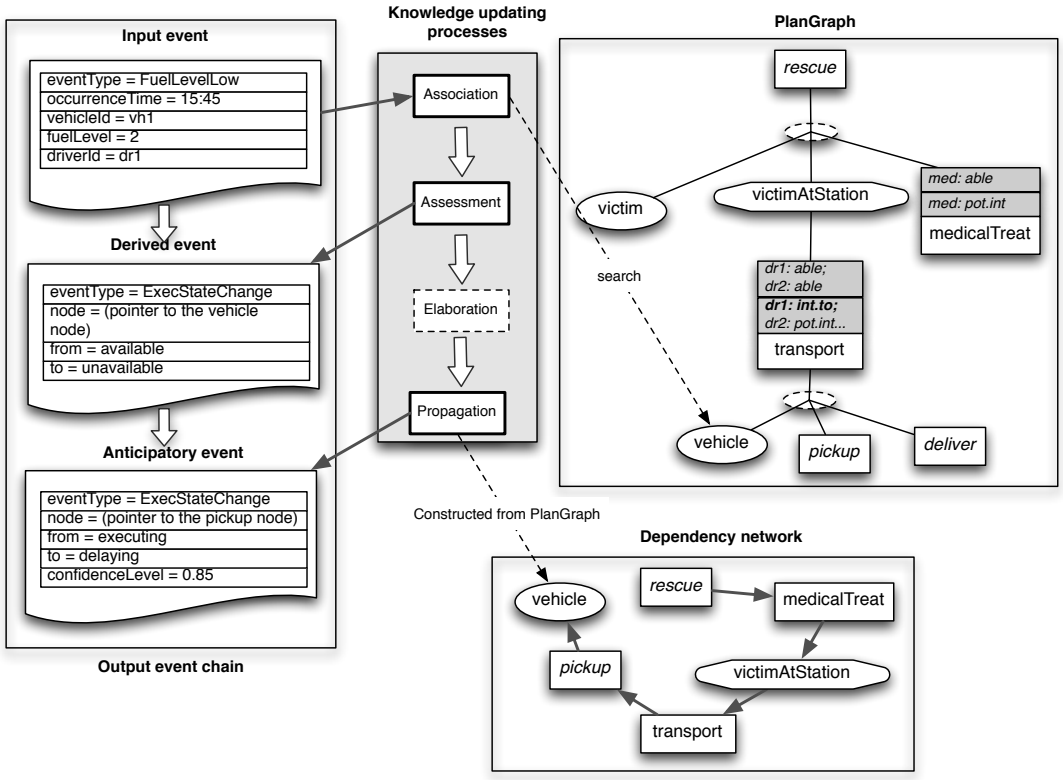


Figure 5.15. The knowledge updating example (*Event 3*)

and dependencies. The PlanGraph model exhibits some important characteristics that make it suitable to satisfy the three requirements for representing collaborative activities described in Section 4.3.3.

1. The PlanGraph model represents collaborative activities as hierarchically structured actions. The components of actions, parameters, and conditions can be used to capture all the entities and the relations between them. With the knowledge represented in the PlanGraph, the local scopes and dependencies can also be easily identified.

2. 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 reflects the current state of the changing activities.

3. Based on the formalization in SharedPlan theory, the PlanGraph model can be computationally implemented, and it 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 event types 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 collaborative activities. On the other hand, the PlanGraph model enriches the events with more meaningful contextual information, which is used in next chapter to support event-driven awareness processes.

Chapter 6

Promoting Event-Driven Awareness

In this chapter, we discuss our approach to promoting event-driven awareness processes by making use of the knowledge representation of collaborative activities and events. 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 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 emphasize how the computer's knowledge representation of collaborative activities and events can be utilized to reduce complexity and handle dynamics in the corresponding awareness processes.

6.1 Event notification mechanism

Event notification mechanisms have been widely used in event-based awareness systems to support event perception [66]. 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 human actors' interpretations necessary to provide awareness, multiple notifications about the same 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 notified, 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 registers the interest in receiving certain kinds of notifications by submitting subscriptions to the awareness system. The system filters incoming events based on the subscriptions and delivers those notifications that match the user's subscriptions.

As we discussed in Section 4.2.2.1, 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 subscriptions. On the other hand, content-based mechanisms are highly expressive, but require 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 scope 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.

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

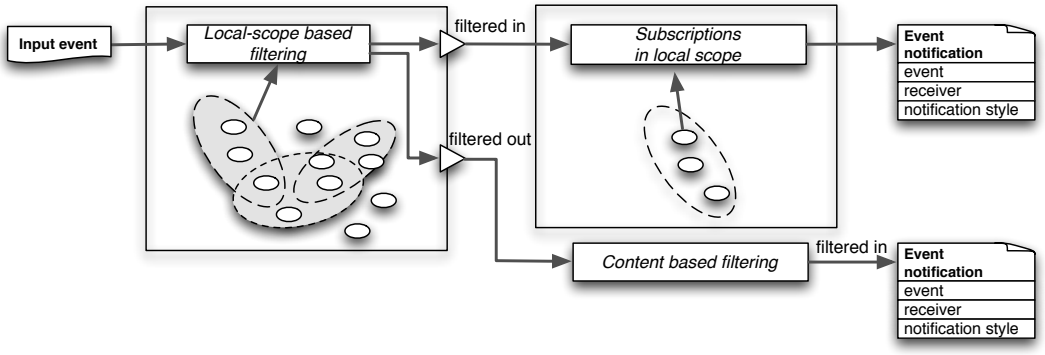


Figure 6.1. Local scope-based event filtering

local scope as the actions that an actor imposes certain level of intention or capability, it is likely that it will be modified as the actor works on different actions. The knowledge updating process as we describe in Section 5.3 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 5.3, not all the events can be successfully associated with entities in the PlanGraph model 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 by the system later. For these events, we should 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 the PlanGraph model, the overhead on the users to manage them is minimized.

In the following of this section, we provide more details on the local scope-based filtering and subscription mechanism. We first describe the algorithm to perform the local scope-based filtering, 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 [68] 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 5.3. 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 events 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 5.1.3.

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 be $TRUE$, and fails the filter if the predicate evaluates to be $FALSE$. Within an actor's local scope, three kinds of filter expressions 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 be $TRUE$ 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*.

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 assign 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 level [80]. Generally, we can define 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 that is affected by the event. This provides less obtrusiveness to the user but is not perceivable unless the entity is within the user's view port.

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.

The computer system can provide interactive tools to support the management of subscriptions within local scopes. We describe an example of the interface to manage subscriptions in our prototype system EDAP and demonstrate its utility in Section 7.3. As the system maintains the knowledge representation of the collaborative activity

and each user's local scope, it is easy to generate a diagrammatic representation of the collaborative activity 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.

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:

On one hand, the system can perform the reasoning tasks for the user. Actually, as we can see in Section 5.3, 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 collaborative activity. In the *propagation* step, the system predicts the future states on other activities. Hence, the system can support the user by simply present these 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 to 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):

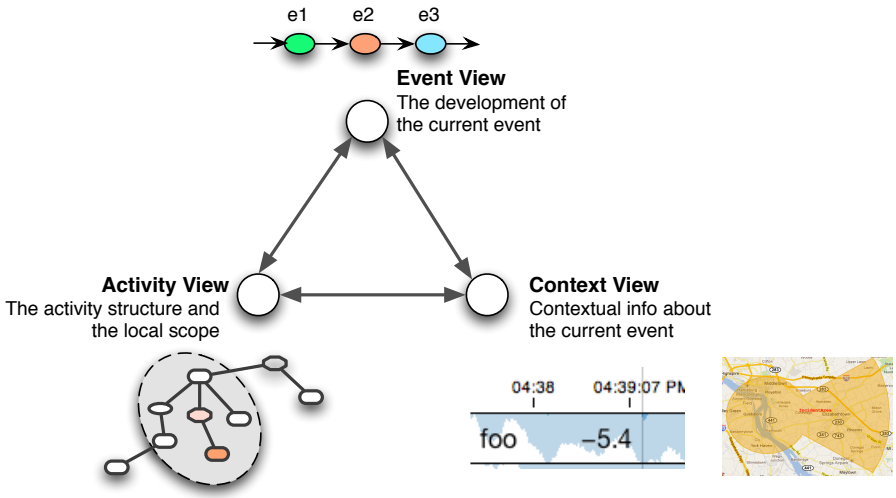


Figure 6.2. The visualization framework for event interpretation

1. Event view: visual representation of the event that needs to be interpreted.

2. Activity view: visual representation of the activity structure 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 5.3, 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 collaborative activity has been changed due to the original event. In the propagation step, the system predicates the future state changes, and adds 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 developmental 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:

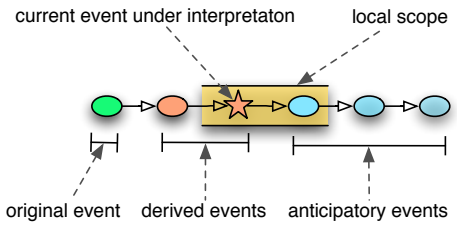


Figure 6.3. Visualizing the event chain in the event view

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 users 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 developmental trajectory of an event across multiple actors.

6.2.2 Activity view

The activity view provides an overview of the activity structure and the user's local scope. Such a representation provides the information about the actions, resources, and actors that are participated in the collaborative activity and the relations between these different entities. As argued by Carroll et al. [21], 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 users:

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 collaborative activity, 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 knows what part of the collaborative activity he/she needs to focus on.

The activity view consists of a diagrammatic representation of the entities and relations in the collaborative activity, 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 collaborative activity 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 view port, 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 [54]. 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):

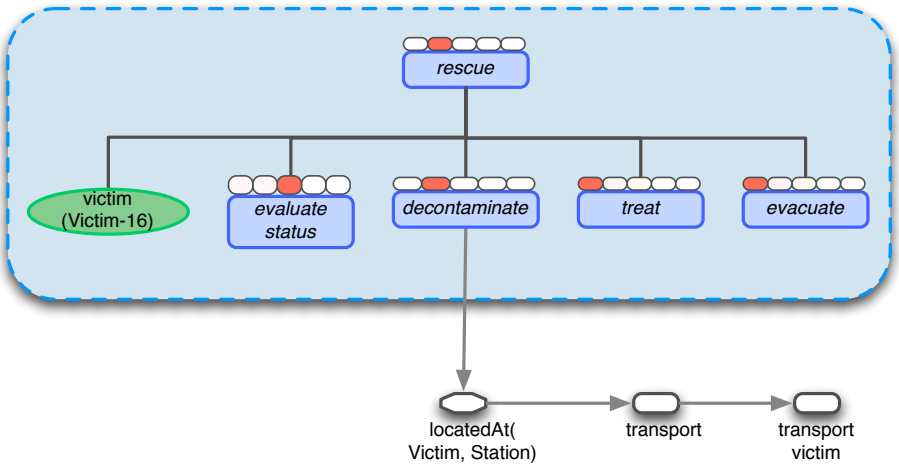


Figure 6.4. An example of the activity view

1. The entities within the local scope are visualized as larger nodes so that they can encode more detailed information, such as the name, type, and execution state. But the entities outside the local scope are visualized as small icons that have less visual information encoded.

2. 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 awareness phenomena in this study. 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 context of an event's origin, or the user's current work context. The contexts emerge from a complex mix of entities in the collaborative activity, 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. The set of views and interactive tools that are needed usually depends on the application domain and each actor's task in the domain. Figure 7.6 in Section 7.3 shows an example of the context view of an victim manager in the emergency response scenario, which includes a bar char to show the victims in different processing stages, a cartographic map to display the distribution of victims, and a data table to list all the victims.

6.3 Mediating event propagation

Event propagation refers to the social process to establish compatible awareness across multiple actors. As we argue in Section 3.3.2, 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 externalizes 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, 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 *actor*1's local scope, hence it will be notified to *actor*1 during the notification process. Upon receiving the notification, *actor*1 performs his/her own interpretation process, and externalizes 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 *actor*2's local scope. As a result, the system notifies *actor*2 about e_2. The process continues as *actor*2 further interprets the event and generates new event, which is later notified to *actor*3.

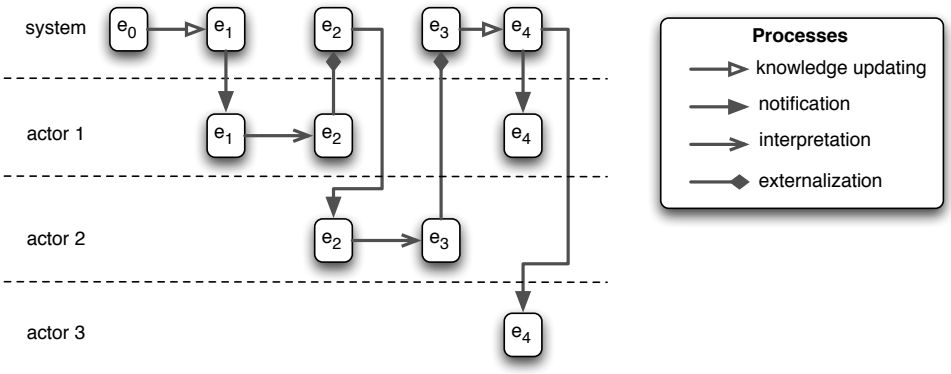


Figure 6.5. An example of the event propagation process

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. On one hand, it relies on

the human actors to interpret the events built on top of each other's interpretation, 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 is enabled by 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.

Beyond that, the system can mediate the event propagation from the human actor's perspective, i.e. to help the human actors to 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 5.3, 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.

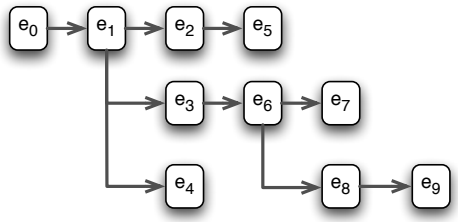


Figure 6.6. The event propagation tree

Similar to the *event chain*, the structure of an *event propagation tree* can be represented by adding additional 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 events 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.

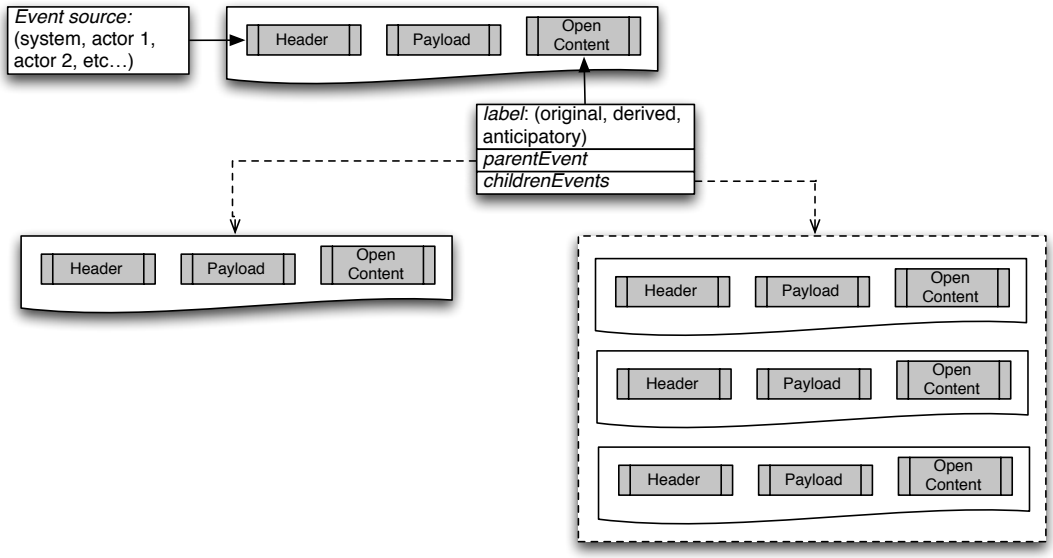


Figure 6.7. Event structure in an event propagation tree

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 [95] 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 group the events 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.

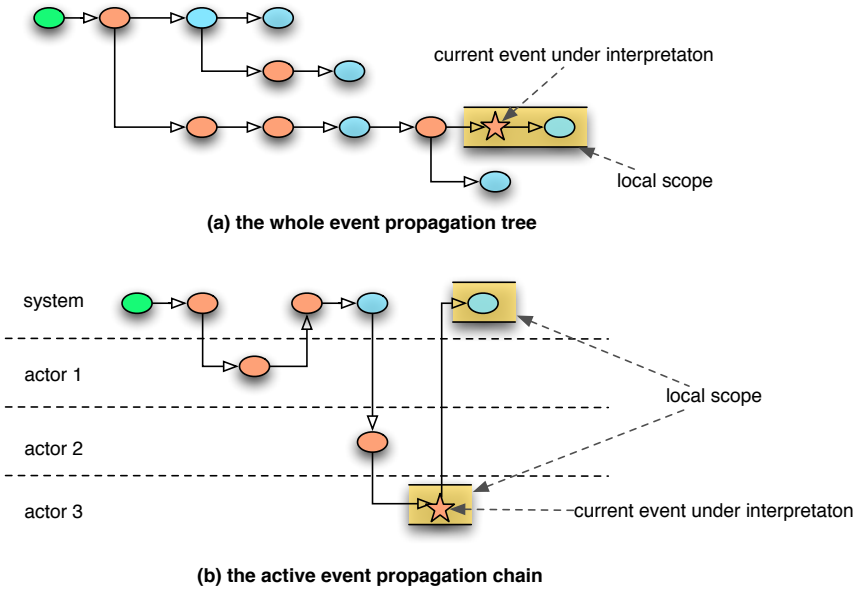


Figure 6.8. Visualizing the event propagation tree in the event view

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 property 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 collaborative activity. As a result, we support the externalization by allowing 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.

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.

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

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.

An example of the event externalization process in our prototype system EDAP is described in Section 7.3.

6.3.3 Controlling visibility

As we allow the human actors to externalize their interpretation as new events, we need to provide the interface for 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 show 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 human actors to contribute. To address this problem, we utilize the local scopes to allow human actors to specify the policies for controlling visibility in an abstract level, similarly as they manage subscriptions for 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 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 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 $APPLY\text{-}VIS$ 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 includes 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 design principle of awareness promotion (Section 4.3) 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 work, 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 contextual 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. Contribute 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 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 recorded by the system, so the human actor does not need to maintain it in the mind. 2. The externalization are directly linked to related nodes.
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. The 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 the 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 [1] 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 [2] is used to offer the push notification from the server to the client by maintaining a long-held HTTP request between them. The EDAP client provides several components that can interact with the server. The *Event*

[1] http://en.wikipedia.org/wiki/Representational_state_transfer
[2] http://en.wikipedia.org/wiki/Comet_(programming)

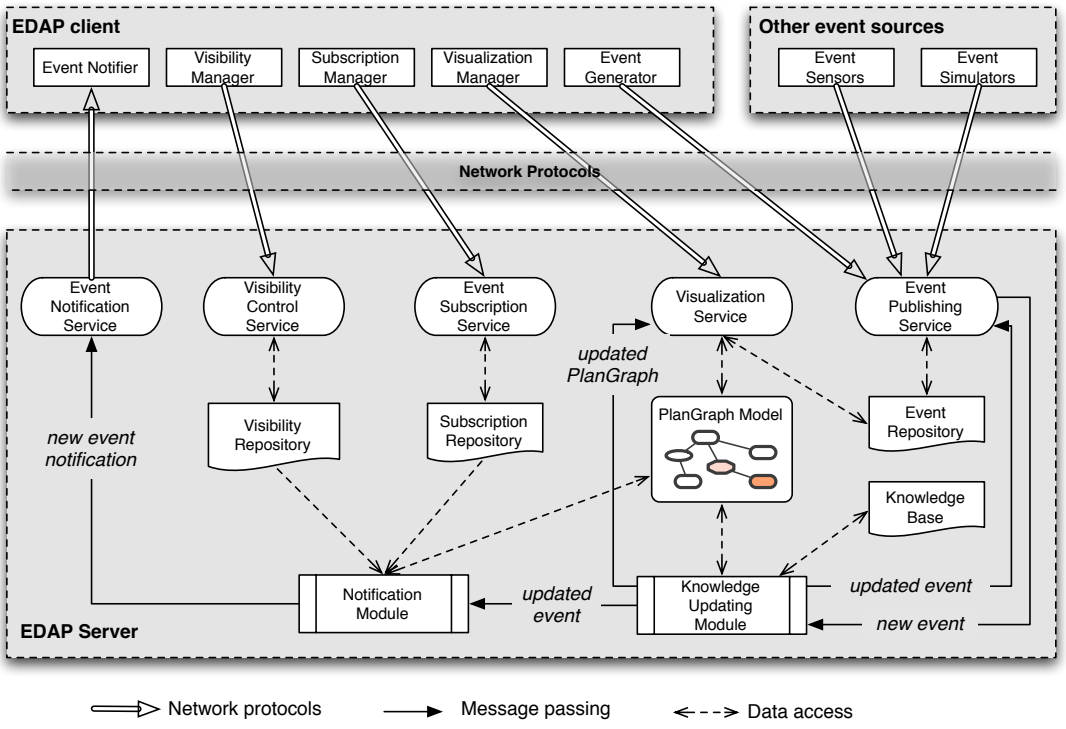


Figure 7.1. Architecture of EDAP

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 5.3 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 that 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 [3] framework. The three data repositories, i.e. event, subscription and visibility repositories, as well as the knowledge base, are implemented as PostgreSQl [4] 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:

[3] http://djangoproject.com
[4] http://www.postgresql.org

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.

Visibility Control Service The *Visibility Control Service* works in the similar way as the *Event Subscription Service*, to manipulate the visibility repository to retrieve,

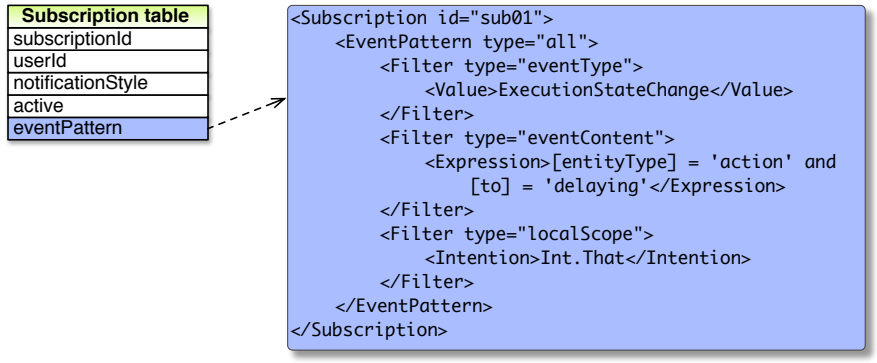


Figure 7.3. An example subscription record

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.

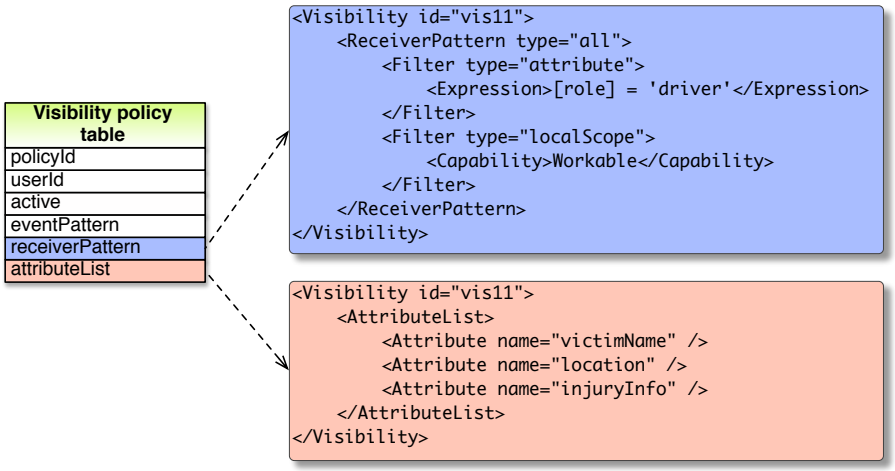


Figure 7.4. An example visibility policy record

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 5.3. 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.

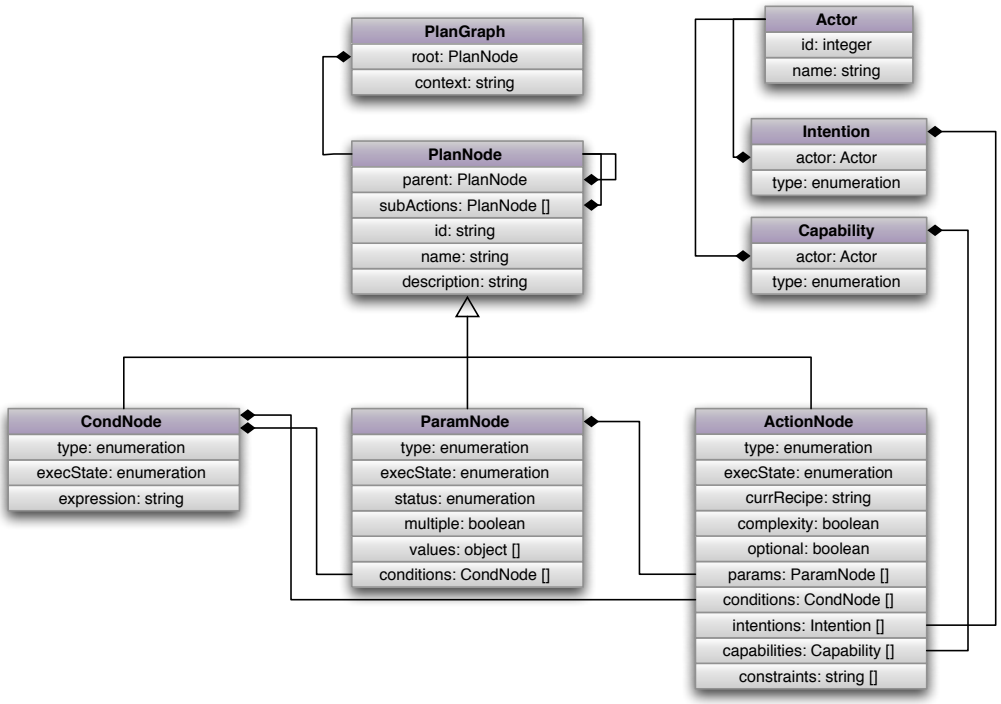


Figure 7.5. The class diagram of the PlanGraph

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 5.3.1. 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 [5] to enable the forward 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 that subscribe to

[5] http://pyke.sourceforge.net

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.

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 5.1.4. 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 [6] framework for the interface layout, the OpenLayers [7] mapping library to provide the geographic context view, and the d3 [8] 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.

To demonstrate the functionalities of the EDAP client, we discuss a concrete use case in the motivating scenario of emergency response. A more detailed description of the scenario will be given in the case study (Section 8.2). In this concrete use case, we consider the user as the victim manager (VM) who is in charge of managing all the victims that need to be rescued. In the normal workflow, VM needs to monitor the status of all the victims. Whenever a new victim is reported by the first responders, VM needs to evaluate the status and decide on whether the victim needs to be decontaminated or treated. For the current victims, VM also needs to make sure the operations on them can be conducted in a timely way. Figure 7.6 shows the main interface for VM during the normal workflow. The interface includes several views to support VM's work.

[6] http://jquery.com
[7] http://openlayers.org
[8] http://d3js.org

1. The *victims* view on the left consists of a set of interactive data visualization tools to explore the victims. For VM, 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.

2. The *victim detail* view shows the detailed information for a selected victim. We provide several context views for the VM to evaluate each victim's information, including the text-based information about the victim, a spatio-temporal trajectory showing the movement of the victim, and a status transition view showing the history of the status change on the victim.

3. The *activity* view shows the current activity structure of the rescue action related to the victim, where VM can interpret the events and externalize the interpretations through direct manipulation.

4. The *events list* view provides a list of current events notified to VM. VM can click on each event to activate the event view to show more detail about the event.

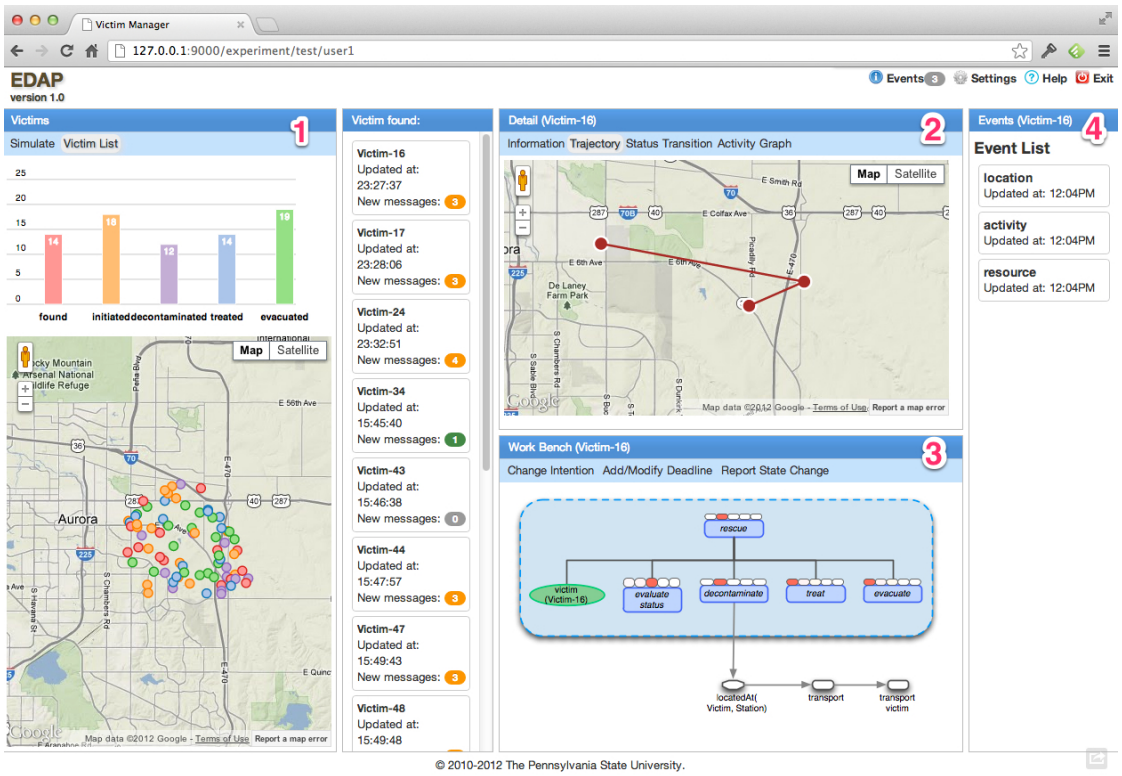


Figure 7.6. The main interface of EDAP for the victim manager
(1)*victims* view, (2)*victim detail* view, (3)*activity* view, (4)*events list* view

Within this use case, we consider the awareness process that is conducted by VM upon receiving an event and discuss the different functionalities provided by the EDAP client to support it.

Event notification The awareness process starts with VM receiving an event notification in the interface. The event $E4$ indicates that the decontamination action on a victim ($Victim$-16) is delayed. Because the decontamination action is within VM's local scope of work, the EDAP server generates an event notification and pushes the notification to VM. As the *Event Notifier* module of the EDAP client receives the notification, it render the notification in the client based on the associated notification style that is defined by VM in the subscriptions. In EDAP client, we support three notification styles with different obtrusiveness levels. At the most obtrusive level, we use an *alert* that is a modal dialog on the interface, blocking everything in the client. The user has to confirm the reception of the notification before resuming the previous state. Then it could be a *banner* that presents each incoming event notification at a dedicated corner for a period of time and then disappears. At the least obtrusive level, it can be a *badge* that displays the total number of events attached with each victim in the victim list. The user can click the badge to open the event list to review the event notifications. The three notification styles correspond to the importance levels when the user manages subscriptions. The events with high importance level will be presented as alerts. Medium importance level corresponds to banners, and low importance level maps to badges. Figure 7.7 shows an example of the three notification styles, and $E4$ in this case is presented to VM as an *alert*.

Event interpretation Upon receiving the notification in the interface, VM can click on the confirmation button in the alert to bring up the event view to perceive more detail about the event. The event view follows the general design principles described in Section 6.2.1. The event $E4$ has been attached with an event propagation tree as it has been developed by several other actors. Hence, the client visualizes the event propagation tree along with the event, so that DM can navigate through event propagation tree to understand the developmental history of the event.

Figure 7.8 shows the active event propagation chain attached with $E4$, which clearly shows the actors that are involved in developing $E4$. DM can switch to the event propagation tree view by clicking the button at the toolbar. With in the event chain, VM can mouse over each node to see detailed information about the event, such as the event type, occurrence time, payroll attributes, the actor who creates the event, and

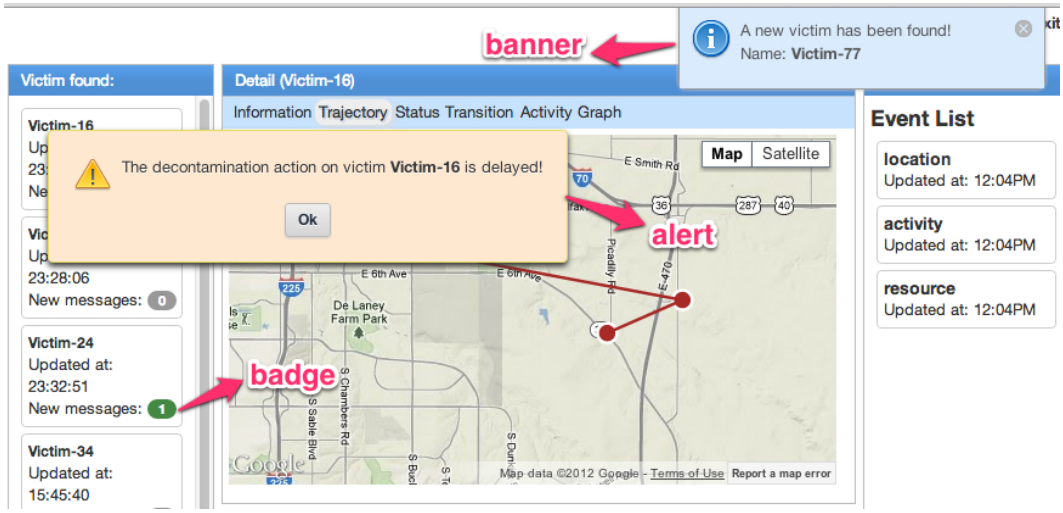


Figure 7.7. Event notification styles in EDAP client

the author's free text notes. When clicking on the event node, the activity view will be adjusted so that the action/resource node associated with the event will be centered and highlighted. By browsing through the event propagation chain, VM understands that the delay of the decontamination action is because of the delay of the transportation action to send the victim to the decontamination station, which is furthered caused by the mechanical problem with the driver's vehicle. In this way, VM can back track to understand the origin of the event.

After understanding the event with the context of origin, VM also evaluates how the event will impact her other actions in the projection process. This is supported by the activity view that is linked with the event view. The activity view provides the activity-related inferential frame for the VM to predict the impacts of the event. Figure 7.9 shows the activity view for VM, which allows VM to navigate through the different entities in the activity structure, check the detail on each entity, and recognize different relations between them. The activity view treats the entities inside and outside VM's local scope differently so that the user always know what part of the collaborative activity he/she needs to focus on, and it also allows the client to provide more detail information for the entities within the local scope. By navigating through the activity view, VM infers that because of the delay of the decontamination action, the rescue action that depends on it will also be delayed.

Event externalization The result of VM's interpretation leads to the further development of the event, as VM believes that the rescue action will be delayed due to the

Figure 7.8. Event view in EDAP client

dependency between decontamination and rescue action. To externalize this belief about the execution state change on the rescue action, VM can click on the node in the activity view representing the rescue action and choose the option to change the execution state. The client shows up an pop-up window that allows the VM to externalize the results of her interpretation, which is directly associated with the rescue action. Figure 7.9 shows the externalization interface where VM can select the expected state for the action, and provides the confidence level and explanation on the interpretation. Whenever VM presses the 'Change State' button, a new internal event representing the VM's belief about the execution state on the rescue action will be sent to the server, with reference to the event under interpretation ($E4$).

Event subscription/visibility Besides the support for event notification, interpretation, and externalization, the EDAP client also provides two supplementary interfaces to support the management of subscriptions and visibilities. The interface to control event visibility is very similar to the event subscription interface, as a result we focus on the event subscription in the following.

Figure 7.10 shows VM's interface to manage subscriptions. On the left is the vi-

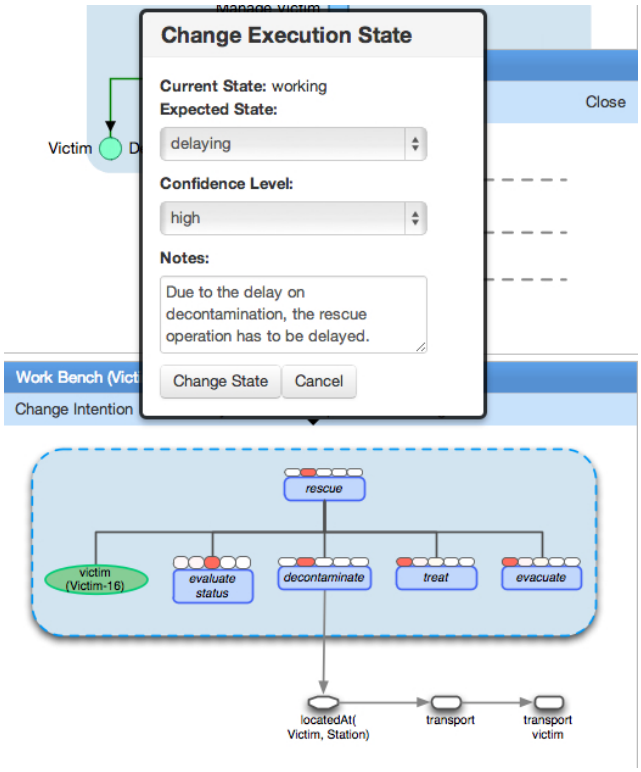


Figure 7.9. Event externalization in the activity view

sualization of the dependency networks along with the different actors' local scopes color-coded. The visualization is presented to the user gradually. In the beginning, only the entities within VM's local scope of work is visible, so that VM can focus on manage the subscriptions with in the local scope. Whenever VM clicks on a node that has dependency relation with other entities outside the local scope, the dependency network is expanded to show the related entities as well, and VM can subscribe to events associated with these entities as well. The expansion of the dependency network can be performed iteratively as VM browses through the entities in a chain of dependencies. In this way, the system provides the initial control to limit subscriptions within the local scope, but at the same time allows the user to override the limitation and subscribe to events that are outside the local scope of work.

To make a particular subscription, VM clicks on a node in the dependency network, and all the visible events on the node is listed on the right of the interface. VM can subscribe to any of the visible events directly or add extra filters on refine the subscriptions. In the EDAP client, we support all the three types of filters we discussed in Section 6.1.1, i.e. filters based on event type, content, and local scope. Besides the filters, VM

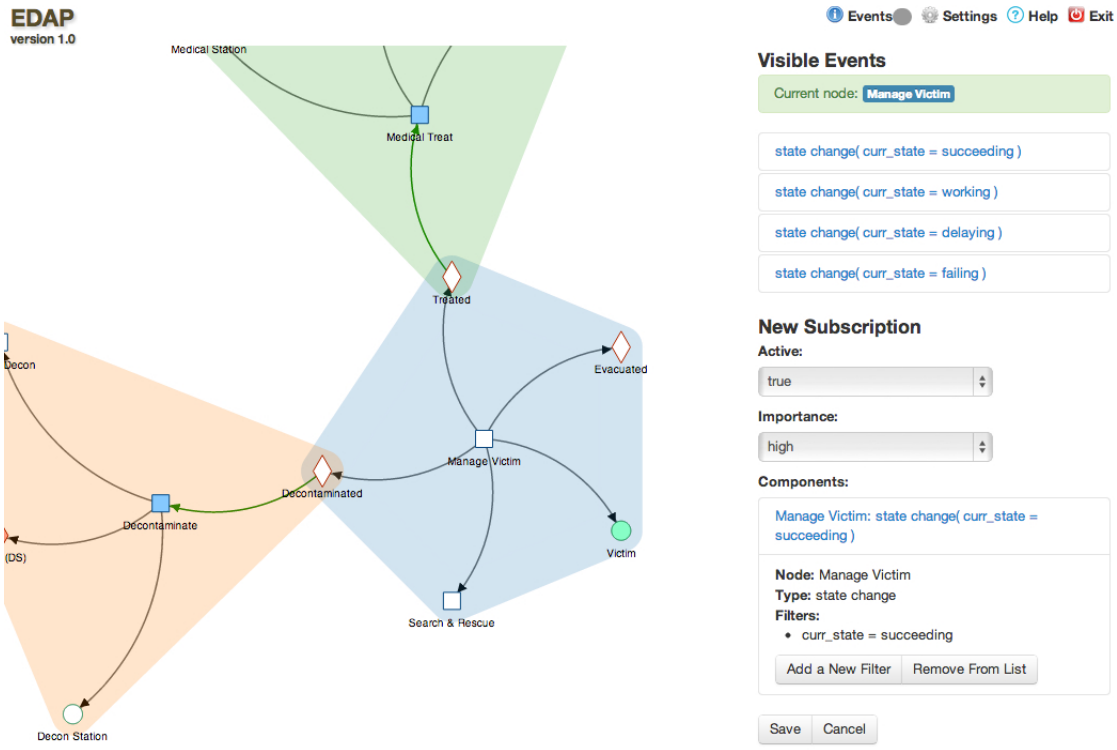


Figure 7.10. Event subscription in EDAP client

can also assign an important level to the subscription, which will be used to determine the notification styles as we mentioned in the event notification step.

7.4 Discussion

A prototype system EDAP (Event-driven awareness promotion) is implemented in this study to demonstrate the feasibility of our computational awareness promotion framework. The EDAP platform includes two components: (1) the EDAP server provides a generic event-driven infrastructure for awareness promotion that can be applied in different application domain. (2) the EDAP client is a standalone Web-based interactive awareness system that is used to demonstrate the functionalities of our approach.

The EDAP server has a modular architecture. Its core module, the *Knowledge Updating Module*, incorporates the PlanGraph model to maintain dynamic knowledge about the collaborative activities and perform automatic reasoning to interpret the events. The implementation of the PlanGraph model and the various reasoning algorithms for knowledge construction and updating shows the feasibility of the knowledge representation and updating component in our approach as we proposed in Chapter 5. The implementation

of the EDAP server also provides us the computational platform to test the validity of the knowledge representation and updating algorithms. We describe such an effort through a simulation experiment in the case study (Section 8.4).

The EDAP client demonstrates the functionalities of the approach in supporting different awareness processes.

1. The client supports the event notifications based on the local scopes, and manages multiple notification styles in event subscriptions.

2. It supports the backward tracking to understand the origin of an event by visualizing the event propagation tree in the event view.

3. It supports the forward tracking to evaluate the potential impact of an event by providing an adaptive visualization of the collaborative activity. The visualization in adaptive in the sense that it provides more detail for the entities within the actor's local scope of work and is dynamically changed as the model of the collaborative activity is changed.

4. It allows the users to directly externalize their interpretations in the activity view, which provides an efficient way to create new events that are associated with internal aspects of the actions and linked to other events in the developmental trajectory.

5. Lastly, it provides the functionalities to manage event subscriptions and visibilities, which allows the users to retain controls in the awareness processes and customize the computer system's behaviors.

Chapter 8

Case Study: Promoting Awareness in Emergency Response

This chapter demonstrates the utility of the event-driven awareness promotion approach through a case study in the context of the emergency response [1]. 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 five 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 external events involved in the process. We develop four episodes in the scenario to demonstrate the typical development trajectories that may happen in collaborative activities. This 'problem scenario' [86] describes the current practices in the emergency response operations and sets up the grounded unit of analysis for our computational approach.

3. The third step consists of an analysis of the awareness phenomena in the scenario based on our conceptual model in Chapter 3. The analysis shows the capability of using the conceptual model to understand the major characteristics and developmental trajectories of collaborative awareness in the scenario.

[1] Partially result of this case study has been reported in a recent paper in ISCRAM 2012 conference [117].

4. Next, we conduct a computational simulation of the four episodes in our prototype system, the EDAP platform to examine the expressiveness of the PlanGraph model in our approach to represent the collaborative activity and validate the knowledge updating process.

5. Last, 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. We demonstrate the key strengths of our approach in several concrete use cases, as it would be if EDAP platform was used to promote awareness in the scenario.

The analysis of the case study includes several design evaluation methods as summarized in [56]. The *analytical* method is used to analyze the use of the conceptual model to understand different characteristics of the awareness phenomena (Section 8.3). The *experimental* method is used to simulate the knowledge updating processes in the four episodes (Section 8.4). The *descriptive* method is used to demonstrate the strength of our awareness approach in detailed use cases (Section 8.5).

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 [94]. 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 [111]. 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 [67]: 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 [111]. 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 [14], and offering better feedback on the appropriateness of actions arising from such decisions [24]. With respect to supporting team-level awareness, research into and development of emergency response support has mainly focused on shared situation awareness [109], 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 [109].

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 [111]. 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 [110]), the limitations of event-based models as we discussed in Section 4.2 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 during awareness transactions.

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 [2], the Nuclear/Radiological Incident Annex to NRF [3], NRC Incident Response Plan [4], as well as a collection of FEMA after action

[2]http://www.fema.gov/national-response-framework
[3]http://www.fema.gov/pdf/emergency/nrf/nrf_nuclearradiologicalincidentannex.pdf
[4]http://www.nrc.gov/about-nrc/regulatory/incident-response.pdf

reports related to specific emergency exercise [5]. To simplify the analysis, however, we reduce the complexity of the emergency response operations significantly in this scenario, comparing with the real world situations as described in these documents.

In the following of this section, we first provide an overview of the scenario, with the major elements in the collaborative activity, i.e. the actors, resources, and actions, as well as the external events. Then we describe four concrete episodes in the scenario, reflecting the different developmental trajectories of the collaborative activity.

8.2.1 Overview of the scenario

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 search for victims in the impacted area, (2) decontamination and medical teams to assist victims, (3) transportation team to provide prompt delivery of personnel, equipment, and victims, (4) police to confine population and guide the evacuation, (5) and scientific team

[5] http://www.nrc.gov/about-nrc/emerg-preparedness/related-information/fema-after-action-reports.html

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, and some are shared by multiple actions. For instance, the rescue vehicles can be used for transporting victims, and delivering actors or other resources. The resources can be 'tools' such as the measuring equipment, and also be 'objects', such as the victims that are manipulated by multiple actors in different tasks, such as decontamination, medical treatment, or transportation.

Here we highlight several types of resources that are frequently in the four detailed episodes:

1. The victims are the people in the impacted area that have be contaminated by the nuclear radiation and/or physically injured.

2. In order to perform decontamination operations on the victims, several mobile decontamination stations have been set up, and are managed by a decontamination manager in the EOC. Each station includes several decontamination operators to monitor the resources (e.g. the rinse tanks) and perform the decontamination operations.

3. Similarly, several mobile medical stations are also set up to perform medical treatment on the injured victims. The medical stations are managed by a medical manager in the EOC, and each of them includes several paramedics.

4. Multiple rescue vehicles are available to provide prompt delivery of victims, equipment, and rescue personnel. Each vehicle is associated with a dedicated driver, and is managed by the transportation manager.

Actions The emergency response in the scenario is consisted of a large number of actions. We follow the analysis in [110] 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 rescue team needs to be sent to the victim for immediate treatment.

Events The development of the collaborative activity in the scenario can be driven by multiple external events. Based on the sources where they come from, they can be clarified into five categories:

1. *"Incident"* events represent actual or possible changes of the emergency incident. Events of this type 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.

Type	Action	Subsidiary actions	Involved 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 response 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 radioactivity 2. To measure weather characteristics 3. To provide situational reports continuously	Scientific team

Table 8.1. Actions in the nuclear emergency response scenario

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.

8.2.2 Development of the collaborative activity

An important characteristic of the large-scale collaboration as this emergency response scenario demonstrates is the higher level of dynamics, i.e. the actors work in dynamic

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 radioactivity
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 is called to perform the iodine distribution	Conceptual relation	Officials to plan and control the operational actions
Resources event: Electricity generators will arrive at the station in 1 hour...	Spatio-temporal relation	Transportation team to deliver the equipment
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

setting that entails rapid and frequent changes in environment and activities. As a result, plans of the collaborative activities are under continuous development and the tasks and roles of team members cannot be precisely specified in advance. With the various entities in the collaborative activity and the large number of events that can occur in the scenario, the possible developmental trajectories in an emergency response can be infinite. To simplify the analysis of the scenario, 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 developed over time and we consider several consecutive episodes in the sub-process of rescuing a victim. Each of the episodes represents an example of a typical developmental trajectory of the collaborative activity that serves as the basic unit of analysis in this case study.

Table 8.3 lists the four developmental trajectories that may happen in the collaborative activity, and in the following of this section we describe the corresponding episodes in the process of rescuing a victim.

Episode 1: Goal activation The first episode represents the initial development of the rescue action triggered by an event.

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
Role 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 role 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. Developmental trajectories in the scenario

Episode 1 The fireman arrives at the impacted area and searches for victims. When the fireman finds a victim, she reports the discovery of the new victim as a new event. The event about the new victim is received by the victim manager (VM) and VM activates the goal that the victim needs to be rescued. In order to rescue the victim, the VM first needs to evaluate the status of the victim, and then decides on the future actions on the victim. Based on the evaluation of the victim's status, the VM finds that the radiation level of the victim exceeds the threshold, as the result 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 also needs to be evacuated to a shelter for recovery. VM exchanges the goal to decontaminate the victim with the decontamination manager DM. Upon receiving the information from VM, 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. To perform decontamination, DM first needs to assign a decontamination station to the victim. After the assignment of station ($DS1$) to the victim, DM activates the goal to transport the victim to $DS1$, so that the decontamination can be performed. DM exchanges the goal to transport the victim to the station to the transportation manager TM, who then 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. TM first assigns a driver ($DR1$) with a rescue vehicle to the action, and exchange the assignment to the driver ($DR1$). As the driver $DR1$ receives the assignment, $DR1$ starts to perform the action. Besides, the medical manager MM receives the information from the VM 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 it.

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.

> **Episode 2** The operator at $DS1$, where the decontamination of the victim will be performed, is monitoring the resources in the station and prepared for the arrival of the victim. However, the operator finds that the quantity of an important resource, i.e. the available rinse tanks at $DS1$ is running low. This resource shortage is then reported to DM, who then infers that the $DS1$ cannot work properly to decontaminate the victim if this resource shortage is not resolved. As a result, a new action to resolve the resource shortage is added to the current plan of rescuing the victim. To achieve the goal to resolve the resource shortage, DM first needs to locate an inventory where the rinse tanks can be supplied. After identification of the inventory, DM activates the goal to deliver the resource from the inventory to the $DS1$. The goal to deliver the resource is exchanged to the inventory manager IM. Because the goal falls into IM's responsibility, IM starts a new action to deliver the resource. 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 exchange the goal with TM. Upon receiving the information from IM, 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 exchange the assignment to $DR2$ who starts to perform the transportation action.

Episode 3: Role transfer Episode 3 starts with an event leading to a breakdown in performing a subsidiary action, which can cause problems on the whole rescue action. To fix the problem, some actor has to transfer the role and perform an action that is out of pre-defined responsibility.

Episode 3 On the way to pick up the victim and then send the victim to $DS1$, the driver $DR1$'s vehicle encounters a mechanical breakdown. The driver reports the problem to TM. When TM receives the information, TM infers that because of the vehicle breakdown, $DR1$'s action to transport the victim cannot be achieved. To solve the problem, TM attempts to assign another available driver with rescue vehicle to perform the action. However, because all the drivers are currently in duty, TM cannot find one to perform the action until a later time. As a result, TM believes that the action to transport the victim has to be delayed. TM exchanges the information with DM, who further infers that because of the delay to transport the victim, the action to decontaminate the victim will also be delayed. DM exchanges the information with VM, and VM further infers that because of the delay to decontaminate the victim, the action to rescue the victim is delayed. VM exchanges the information with the fireman who is with the victim in the field. The fireman infers that the reason for the delay to rescue the victim is because of the original problem to delivery the victim. As a result, the fireman believes that even though the delivery of the victim is not usually in the scope of her responsibility, she can help the who team by sending the victim directly to $DS1$. As a result, the fireman activates the goal to send the victim to $DS1$, and exchanges the goal with the team. Upon receiving the information 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 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.

Episode 4 As the fireman decides to directly send the victim to DS, 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, which is received by 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 rescue team needs to be formed and dispatched to the victim's location so as 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 exchanged to relevant actors. Upon receiving the information from the VM, DM first drops 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 exchange the assignment of the operator to the team. Similar to DM, MM first removes the action to treat the victim from her current responsibility, and decides on the paramedic that will be assigned to the rescue team. Then the TM infers that in order to perform the action to dispatch a rescue team to the victim, the goal 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, $DR2$ who was assigned for delivering the equipment during the decontamination action is now freed up and is available for this action. As a result, TM exchanges the new assignment with $DR2$ and starts to perform the action.

8.3 Understanding the awareness phenomena in the scenario

In this section, we use the four episodes as described in the problem scenario to understand the awareness phenomena based on our conceptual model. The analysis includes two parts: (1) a goal-directed task analysis to identify the major entities, local scopes, and dependencies in the collaborative activity, which provides the basis for to understand the characteristics of awareness in the scenario; (2) an interaction analysis to identify the awareness transactions in each episode, which shows the idea of how collaborative awareness is socially constructed through multiple awareness transactions.

8.3.1 Analyzing the characteristics of awareness

To identify the major entities local scopes, and dependencies in the collaborative activity, we conduct a goal-direct task analysis following the principles described in [31] for each

episode. However, unlike the original approach that only focuses on the decomposition of the goals into sub-goals, we are also interested in identifying the local scopes and dependencies between actions. As a result, the cognitive task analysis performed in our study includes several steps:

1. For each episode, we first identify the major entities, i.e. the actors, resources, and actions.

2. Then we identify the set of relations between these entities that reflect the common characteristics of the activity structure as we described in our conceptual model. This includes the intention and capability relations between actors and actions, the decomposition and precedence relations between the actions, and the resources that are manipulated or used by the actions.

3. Based on the relations identified in previous step, we construct the local scope of each actor by filtering out the actions that have certain relations with the actor.

4. Last, we look into the relations between actions and relations between actions and resources to identify all the temporal, resource-related, and goal-related dependencies among the actions.

By performing the cognitive task analysis, we are able to construct the local scopes and dependency networks at the end of each episode.

Figure 8.1 shows the local scopes of the decontamination manager DM in the four episodes. From Episode 1 to Episode 2, DM's local scope has been expanded as the action to resolve the shortage has been added and elaborated. From Episode 3 to Episode 4, however, the actions in the DM's local scope are totally different due to the re-planning of the rescue action in Episode 4.

Figure 8.2 shows the dependency network between actions in the collaborative activity after Episode 1. Similarly, we can construct the dependency networks for all the other three episodes.

We use these constructed local scopes and dependency networks to analyze the three essential characteristics of the awareness as we described in our conceptual framework (Section 3.2).

8.3.1.1 Partiality of individual awareness

The first characteristic of awareness in our conceptual model is the *partiality* of individual awareness. The idea is that, since each actor only engages in a small set of actions in

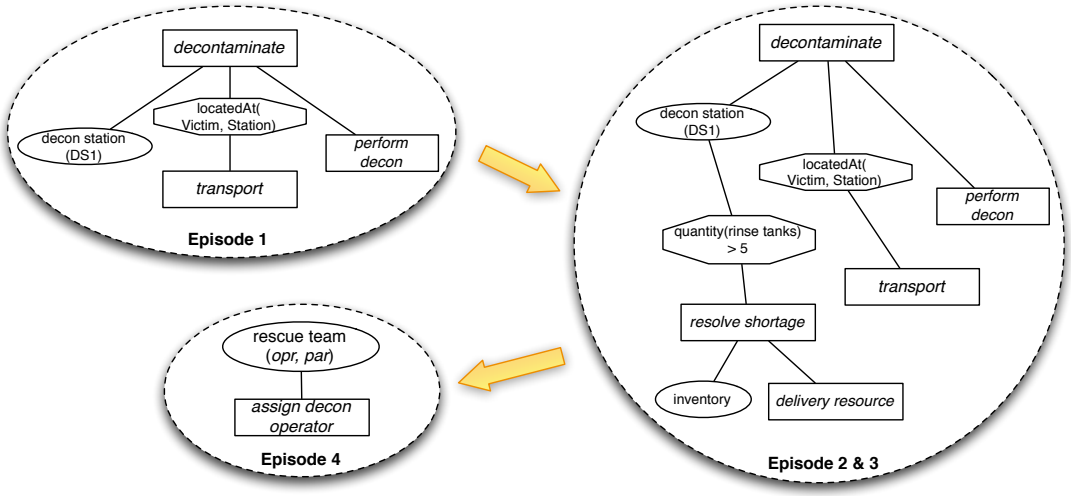


Figure 8.1. The local scopes of DM in four episodes

a large-scale collaborative environment, the actor does not need to fully understand the whole situation. Instead, the information that an actor should be aware of is only partial as it is related to the actions within the actor's local scope of work.

To understand the partiality of individual awareness, we compare the magnitudes of the whole collaborative activity with the local scopes 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 magnitude is defined as the total number of entities (i.e. actions, resource, and actors) in the local scope or the whole collaborative activity. The chart in Figure 8.3 shows the counting results. As we can see from the chart, the total number of entities in the collaborative activity (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). From Episode 1 to Episode 2, we can see the whole activity can grow significantly in a collaborative activity, but each actor's local scope is relatively small, and hence much more manageable for the actor.

8.3.1.2 Compatibility of awareness

The second characteristic of awareness is that the actors' individual awareness needs to be coordinated so as to achieve the compatibility that is collectively needed for the overall team to perform the collaborative task successfully. The requirement for compatible awareness among multiple actors can be accounted for by two characteristics of the

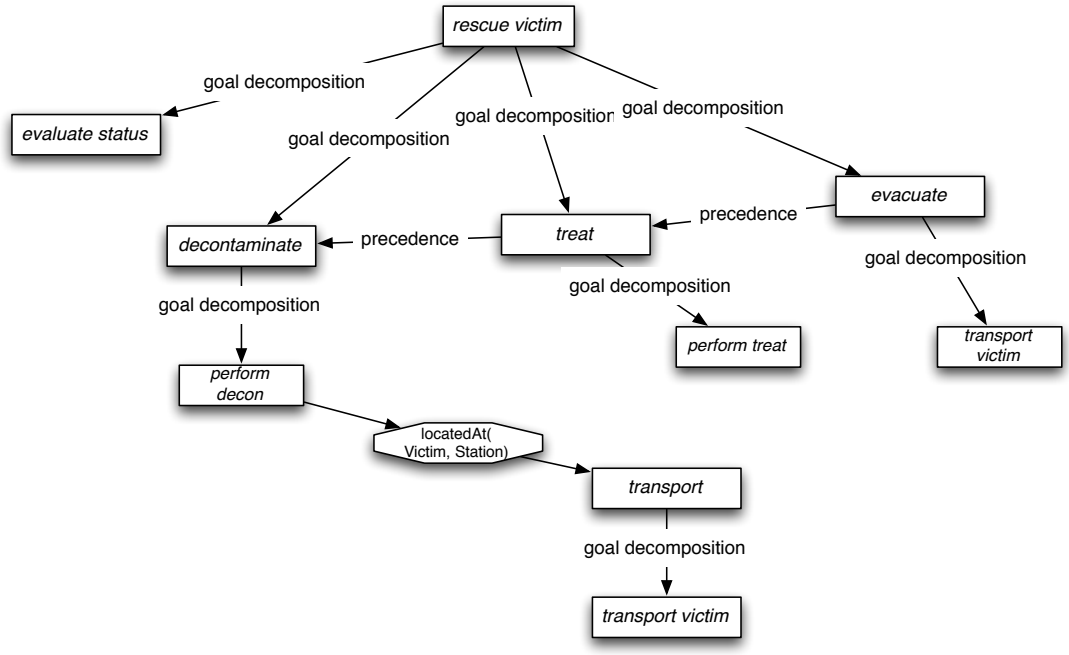


Figure 8.2. The dependency network after Episode 1

collaborative activity: the overlaps between local scopes of work, and the dependencies across local scopes.

To examine the compatibility of awareness, 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 8.4 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 activity after Episode 2 is the one with the highest level of complexity.

From this figure, we can clearly see both the overlaps between local scopes of work, and the dependencies across 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

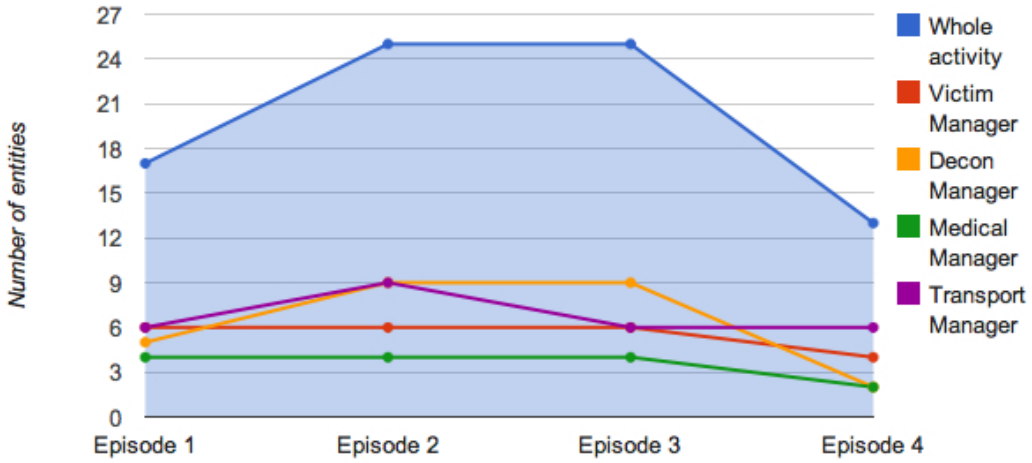


Figure 8.3. Numbers of entities in local scopes

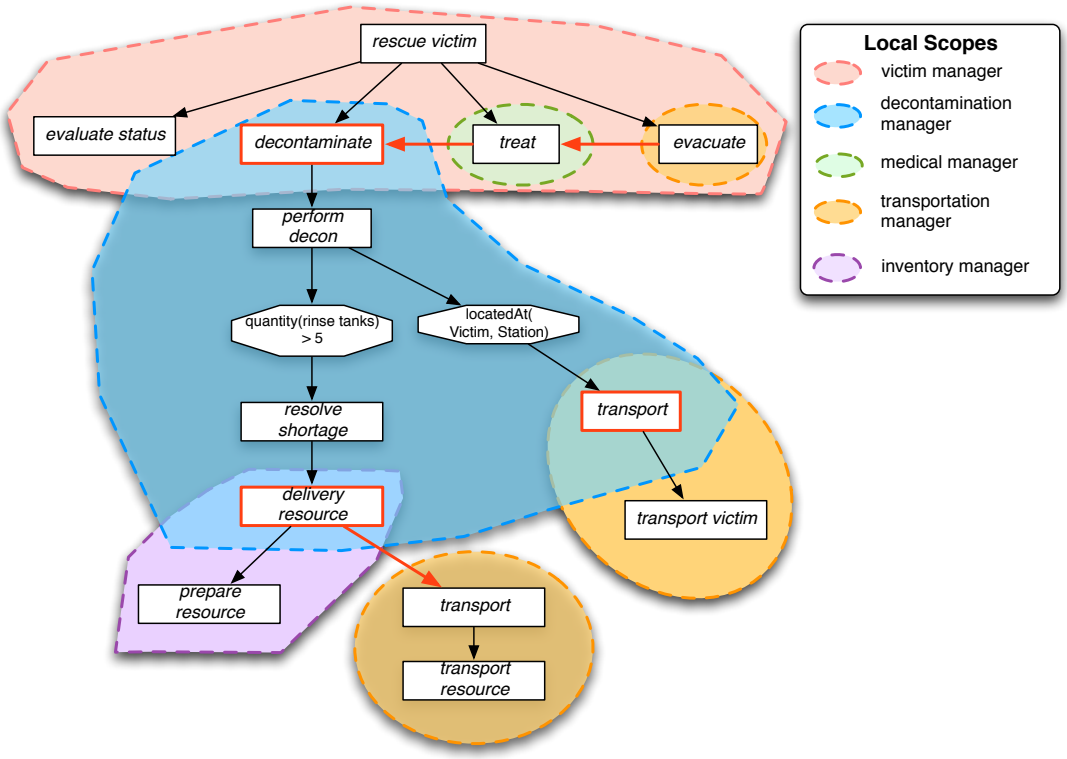


Figure 8.4. Dependency network in Episode 2 with local scopes

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.

8.3.1.3 Dynamics of awareness

Because the local scopes of work, and dependencies are all under continuous change and development in the large-scale distributed collaborative activities, the actors' awareness requirement, i.e. the information an actor is aware of, also entails frequent changes. By comparing the local scopes and dependency networks in each step, we can see that how the local scopes and dependency relations are changing as the collaborative activity is developed.

Figure 8.3 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 8.5 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.

8.3.2 Analyzing the development of awareness

In our conceptual model, we argue that the development of awareness at the team level usually entails multiple transactions through which actors' individual awareness processes are connected as developmental trajectories. To understand the development of awareness in the scenario, we perform a detailed interaction analysis in each episode. The interaction analysis focuses on identifying how the awareness information is passed

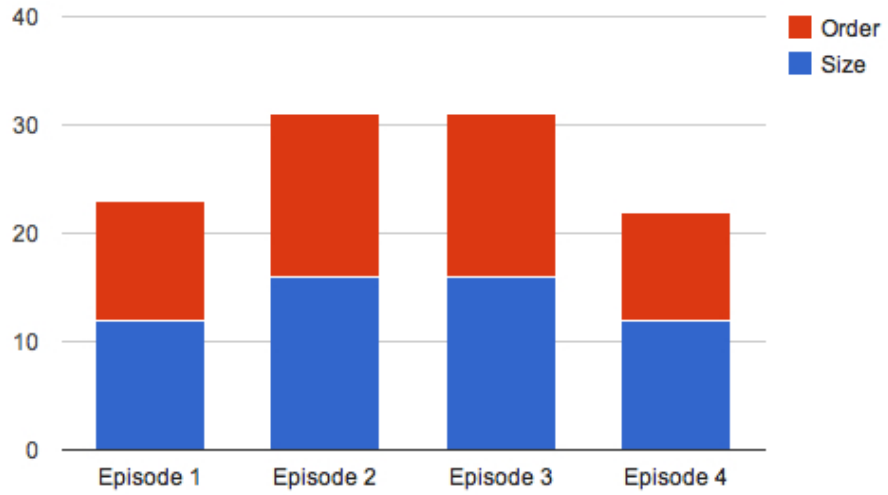


Figure 8.5. Changing dependency networks in the scenario

through the actors in each episode. This section reports the results of the interaction analysis.

8.3.2.1 Episode 1: Goal activation

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 awareness information is generated and more actors are participated into the action. Figure 8.6 shows the developmental trajectory of awareness in Episode 1.

The trajectory starts with the fireman's report on the discovery of the new victim. This initial event is then received by the victim manager VM in an awareness transaction, which triggers VM's individual awareness process. During the individual awareness process, VM activates the goal that the victim needs to be rescued, which leads to the decision to decomposes the action to rescue the victim into sub-actions, and performance of the action to evaluates the status of the victim, which further leads to VM's adoption of the goals to decontaminate, treat, and evacuate the victim. VM then passes the adoption of the goals to the corresponding actors DM, MM, and TM respectively through awareness transactions. Upon Upon receiving the event to activate decontamination, DM starts the individual awareness process that leads to her commitment to performing the action, and further elaborates the action. The results of DM's individual awareness processes generate new awareness information about the activation

of the goal to transport the victim, which is then passed to TM. In a similar way, TM triggers the individual awareness process and passes the activation of the goal to perform transportation task to the driver $DR1$.

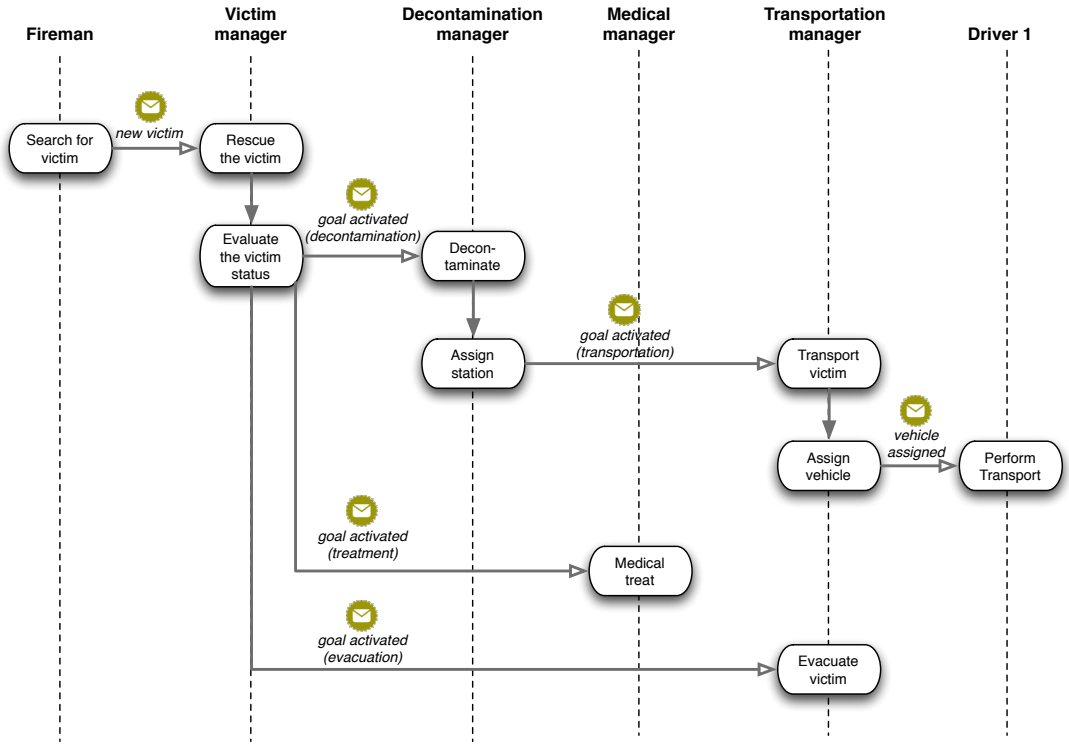


Figure 8.6. Developmental trajectory in Episode 1

8.3.2.2 Episode 2: Plan development

Episode 2 shows an example of the opportunistic plan development during the action performance. Although the action to resolve resource shortage is not a required subsidiary component in the initial plan of 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. However, the developmental trajectory in this episode is very similar to Episode 1 as it demonstrates how the awareness information is propagated as the top-level goal is decomposed into subgoals (Figure 8.7).

8.3.2.3 Episode 3: Role transfer

Unlike the first two episodes, Episode 3 demonstrates the developmental trajectory of the awareness as how the effect of one action can cascade to the other actions because

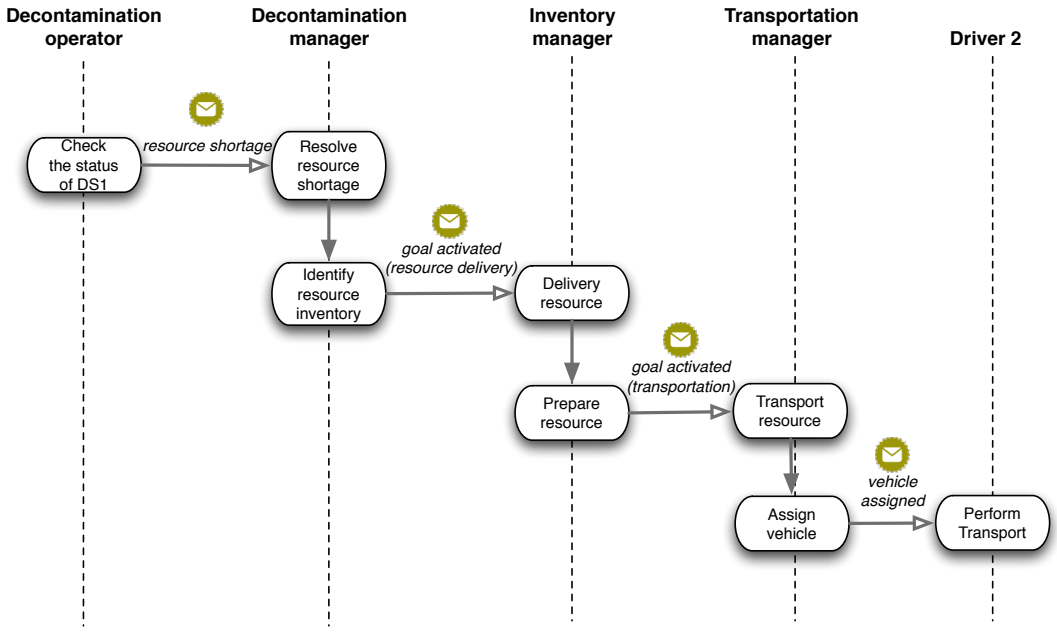


Figure 8.7. Developmental trajectory in Episode 2

of the dependencies among them (Figure 8.8).

Episode 3 starts with the event indicating 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 and passes it to TM. During the individual awareness process, TM first infers that because of the vehicle breakdown, and the fact that all the drivers are currently in duty, the action to transport the victim has to be delayed. TM expresses this interpretation as a new piece of awareness information. Because the DM's action to decontaminate the victim depends on the transportation action, the awareness information generated by TM is passed to DM through an awareness transaction. In the similar way, DM's interpretation on the delay of decontamination action is passed to VM due to the dependency between the rescue action and the decontamination action.

8.3.2.4 Episode 4: Opportunistic re-planning

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, a number of awareness information is derived and distributed around the actors (Figure 8.9).

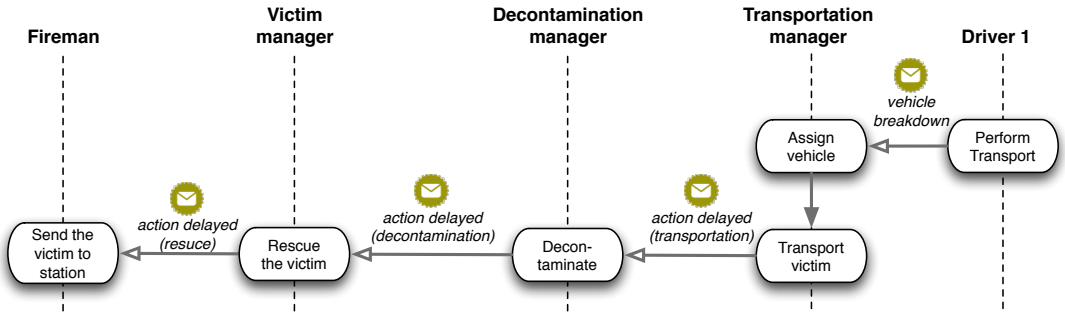


Figure 8.8. Developmental trajectory in Episode 3

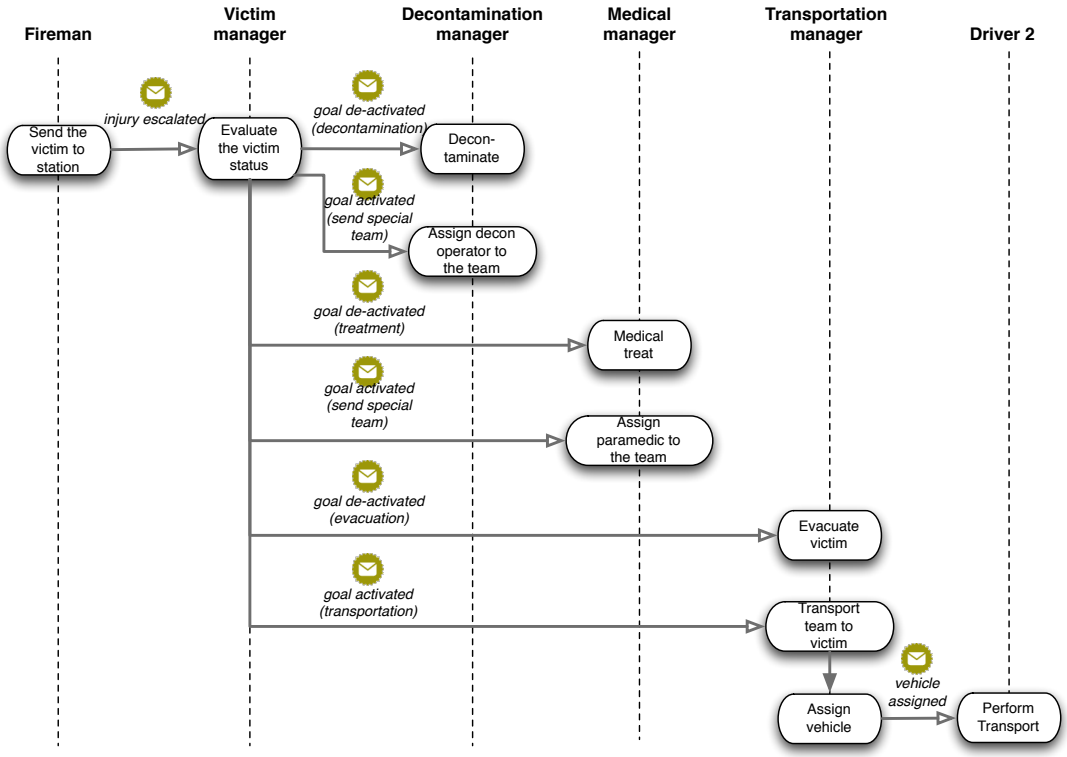


Figure 8.9. Developmental trajectory in Episode 4

8.3.2.5 Discussion

The interaction analysis of the four episodes demonstrates the characteristics of the development of collaborative awareness in our conceptual model:

1. The developmental trajectories of awareness in these episodes clearly show how the awareness knowledge is socially constructed across multiple actors. As the initial event is sent to the team, it is interpreted by an actor who then expresses his/her

interpretation of the event as new awareness information and pass it to other actors through awareness transactions. In this way, the awareness knowledge is undergoing continuous construction as the initial awareness information is interpreted and enriched by multiple actors.

2. These episodes show good examples of how the development of awareness is coupled with the development of the collaborative activity. On one hand are the initial events triggering the development of the collaborative activity, and on the other hand are the changes to the collaborative activity by the actors generate new awareness information. It is within this recursive cycle between the development of awareness and the collaborative activity that the situation is built up.

8.4 Simulating the knowledge updating

The goal of this section is to 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, i.e. how the PlanGraph is updated by the *Knowledge Updating Module*, driven by the set of events occur in the episodes.

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 8.10 shows the two recipes associated with the action to rescue the victim.

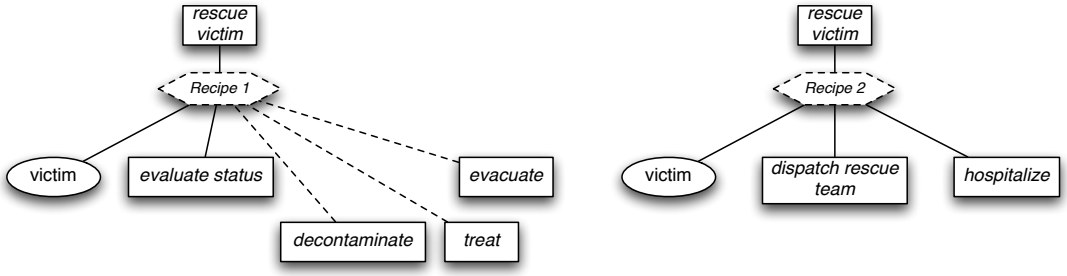


Figure 8.10. Example recipes in the scenario

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. The events are generated based on the interaction analysis in previous section, where each piece of awareness information identified in the interaction analysis is represented as an event. Appendix A shows the four sequences of events generated for the simulation.

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 of the development of the PlanGraph model in the four episodes, which shows the capability of the system to keep track of the event-driven development of the collaborative activity in the scenario.

8.4.1 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 workflow 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 8.11 shows the PlanGraph after the knowledge updating process on the first event ($E1$) in Episode 1.

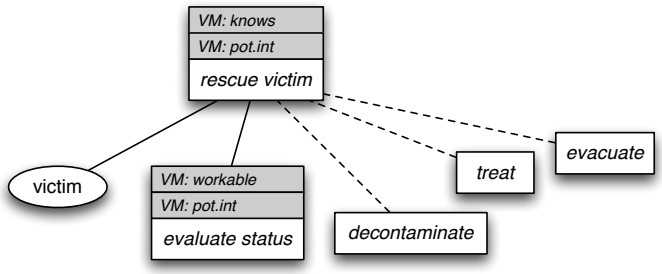


Figure 8.11. PlanGraph after $E1$ in Episode 1

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 actor's 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 VM's 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 8.12 shows the PlanGraph after these events are processed.

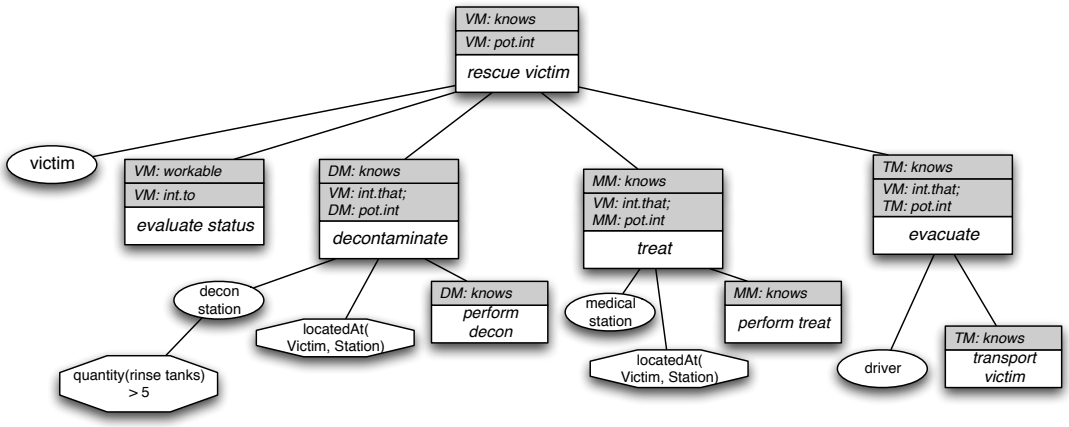


Figure 8.12. PlanGraph after $E4$ in Episode 1

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 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 8.13 shows the PlanGraph after all the events in Episode 1 are processed.

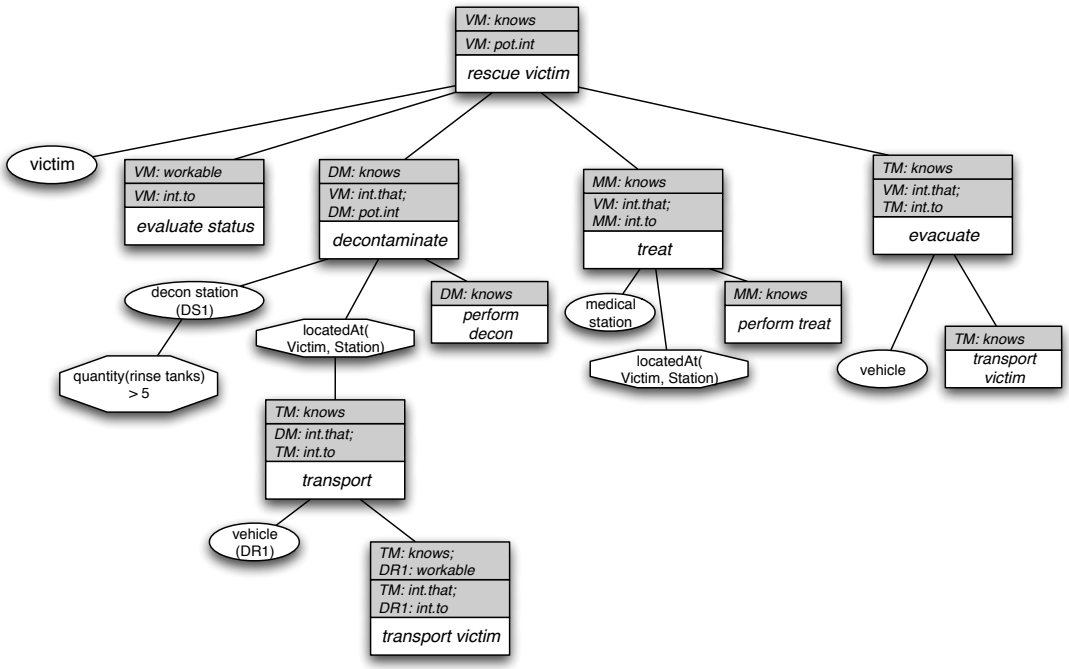


Figure 8.13. PlanGraph after Episode 1

8.4.2 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 8.14 shows the PlanGraph after the knowledge updating process on the first event ($E1$) in Episode 2, and Figure 8.15 shows the event chain generated by the system in the process.

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 elaborates the transportation action. Figure 8.16 shows the PlanGraph after all the events in Episode 2 are processed.

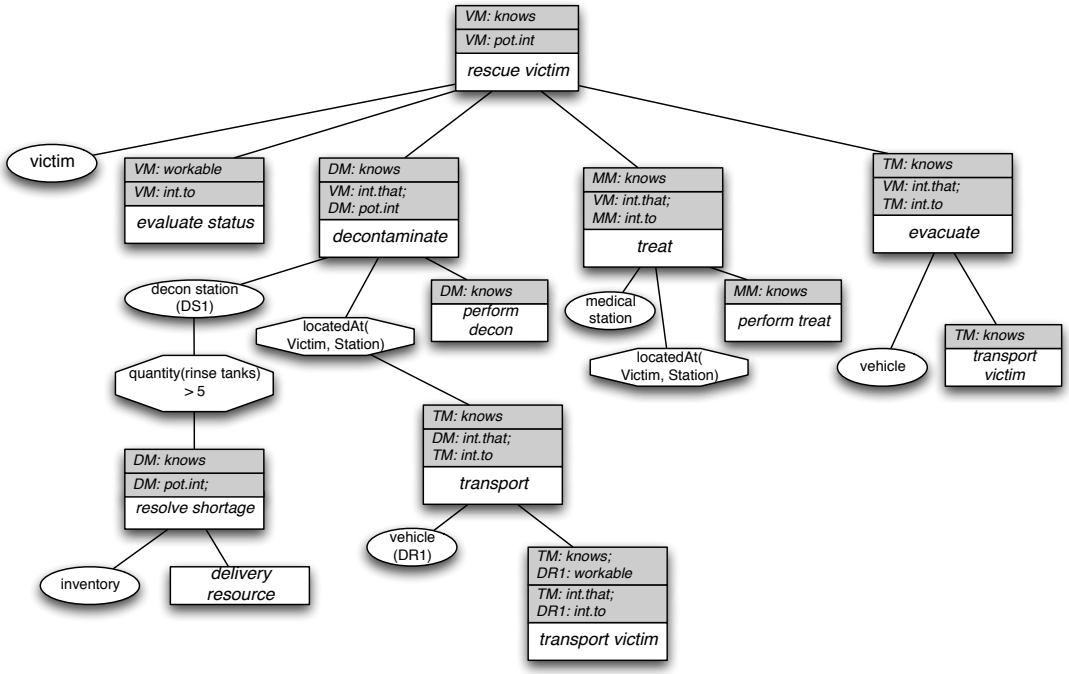


Figure 8.14. PlanGraph after $E1$ in Episode 2

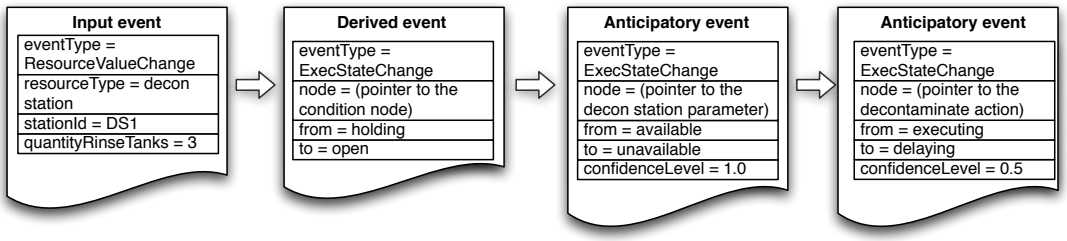


Figure 8.15. Event chain after $E1$ in Episode 2

8.4.3 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

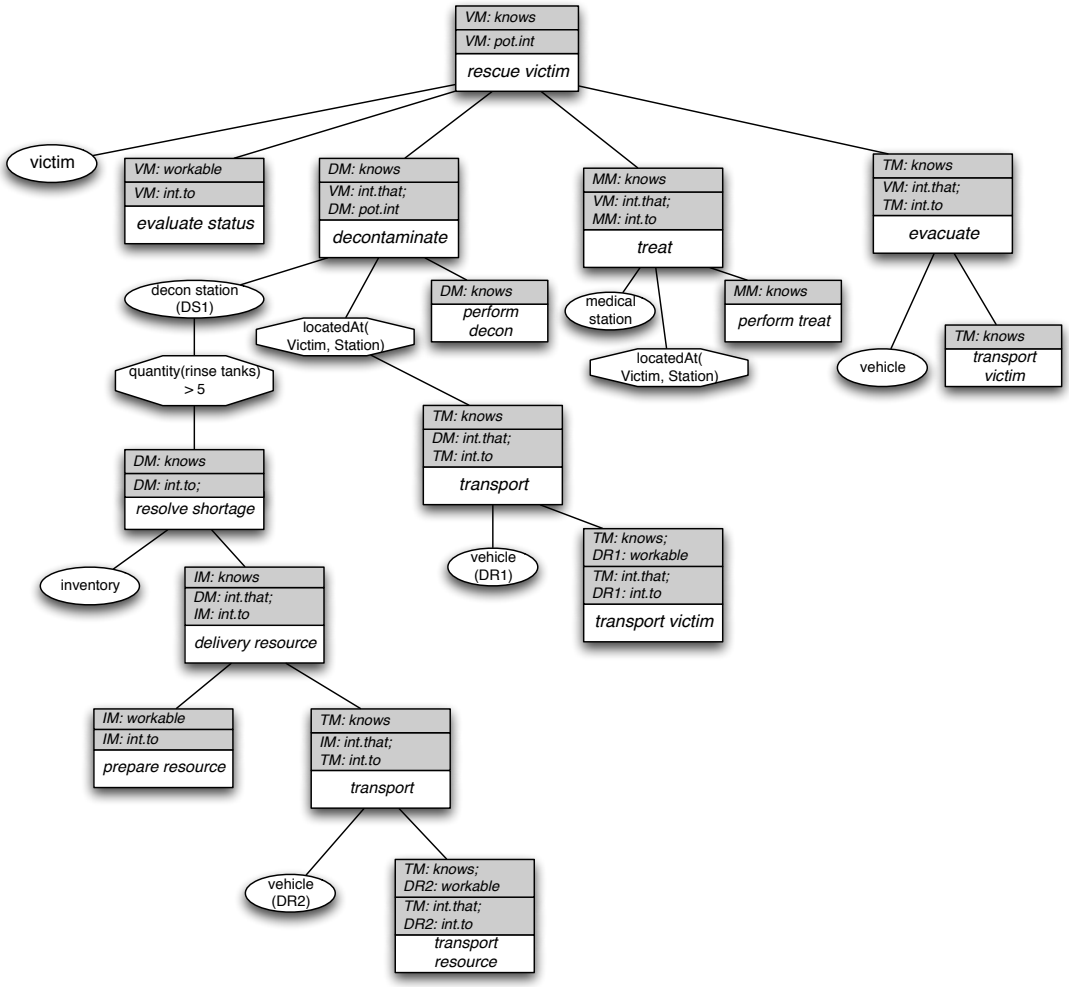


Figure 8.16. PlanGraph after Episode 2

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 8.17): 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.

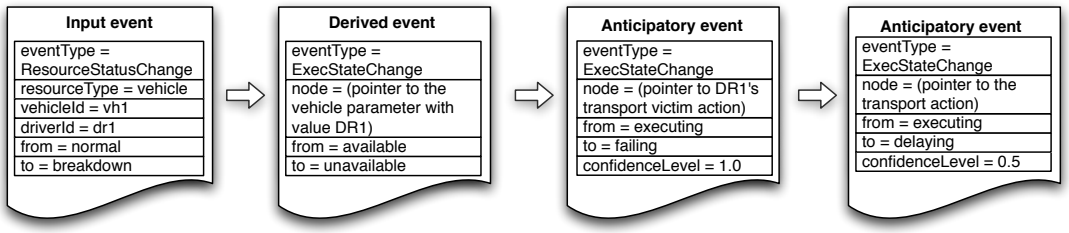


Figure 8.17. Event chain after $E1$ in Episode 3

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 8.18 shows the active event propagation chain that is derived from the event propagation tree, spanning from $E1$ to $E4$.

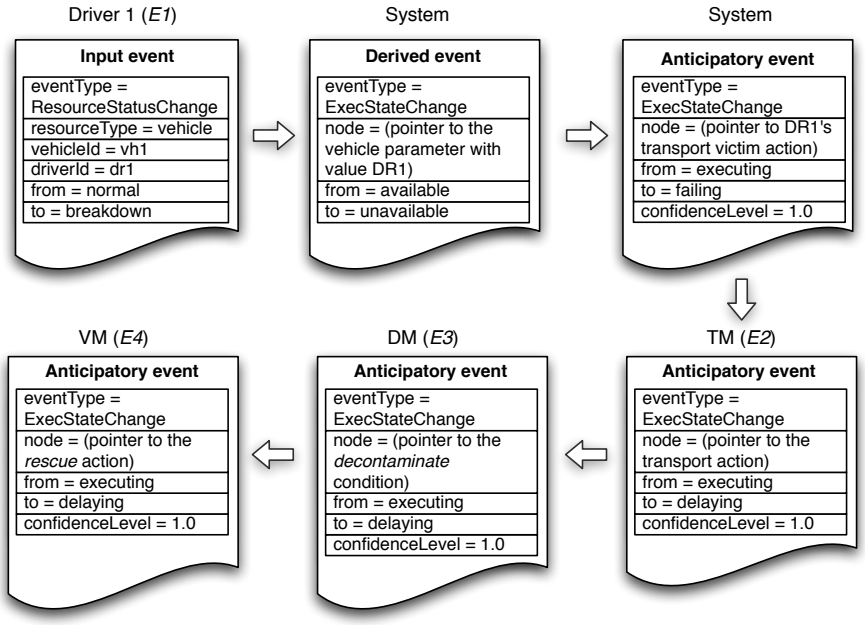


Figure 8.18. Active event propagation chain after $E4$ in Episode 3

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 8.19 shows the

PlanGraph after all the events in Episode 3 are processed.

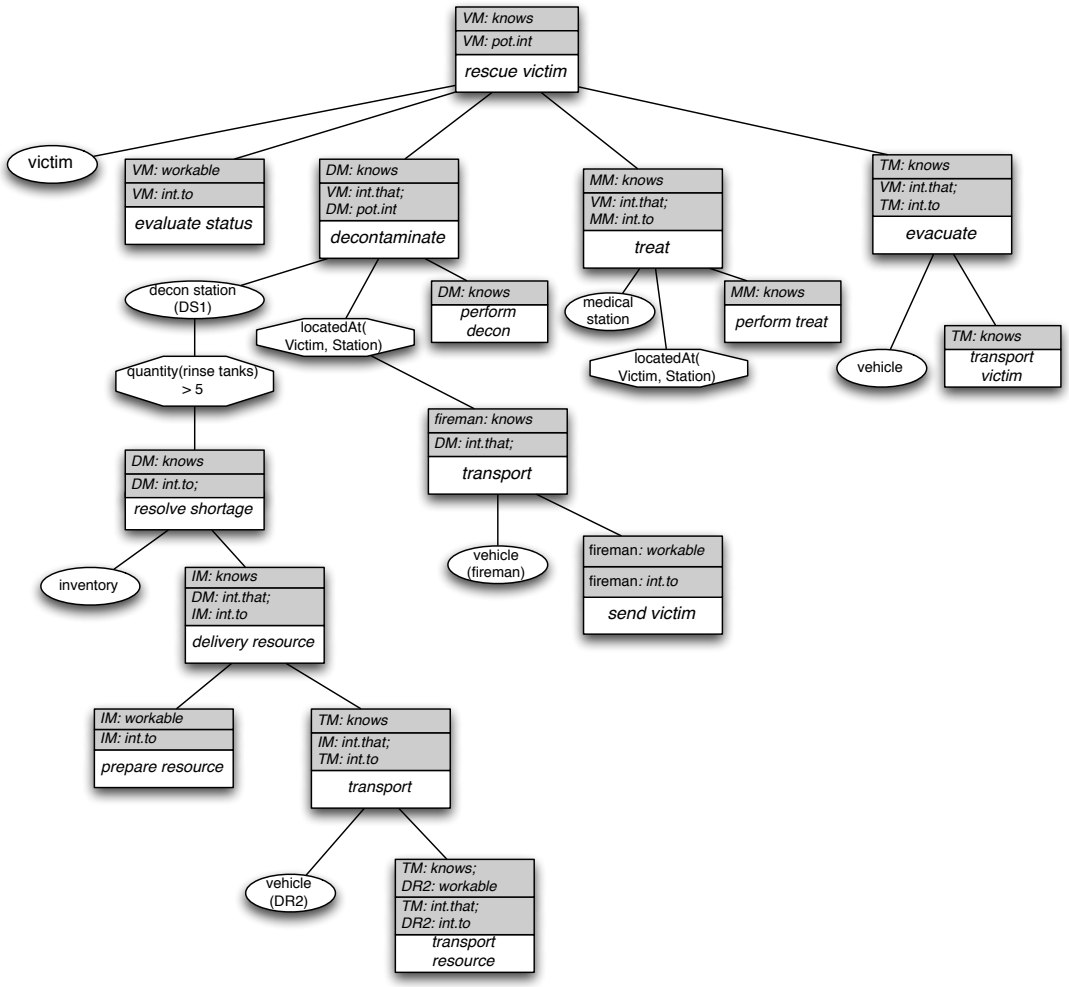


Figure 8.19. PlanGraph after Episode 3

8.4.4 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 updates the attribute storing the injury information. However, such descriptive information cannot be directly 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 applies default knowledge about intentions and capabilities from the knowledge base. Figure 8.20 shows the PlanGraph after the planning events generated by the VM have been processed.

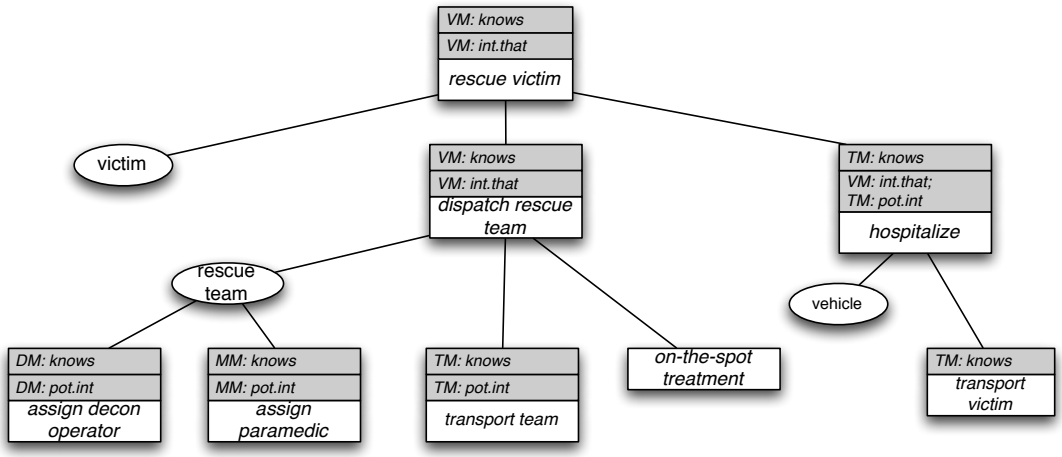


Figure 8.20. PlanGraph after $E6$ in Episode 4

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 8.21.

8.5 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:

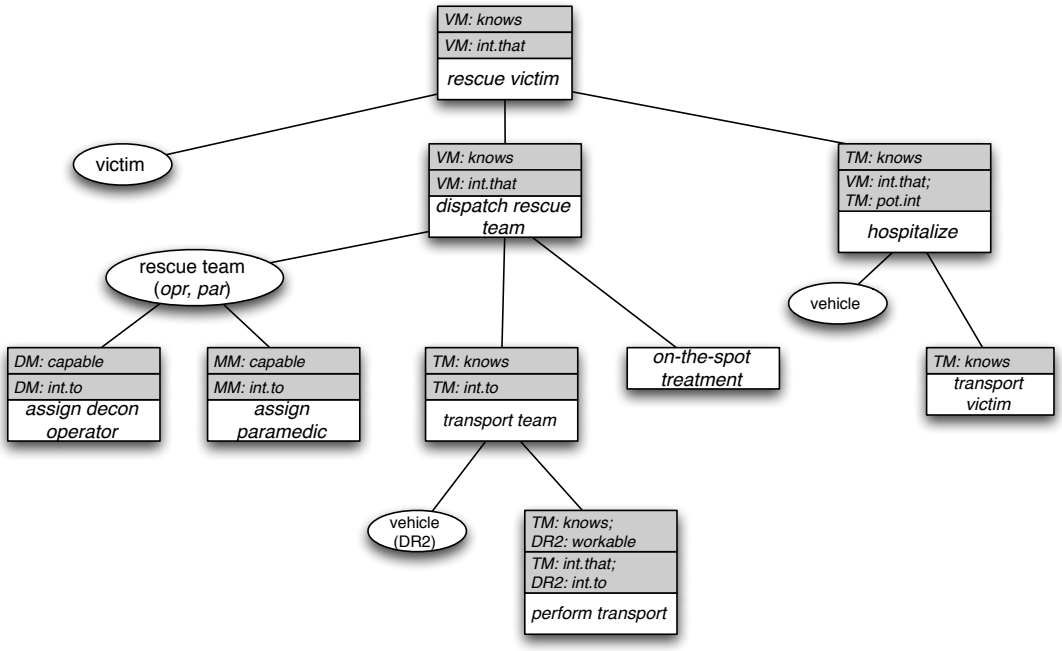


Figure 8.21. PlanGraph after Episode 4

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 compares it with existing awareness support mechanisms.

8.5.1 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 4.2 , 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	**Receiver**	**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 *'rescue'* action. The detailed explanation of this knowledge updating process can be found at Section 8.4.1. 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 8.22 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.

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 8.23 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.

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.

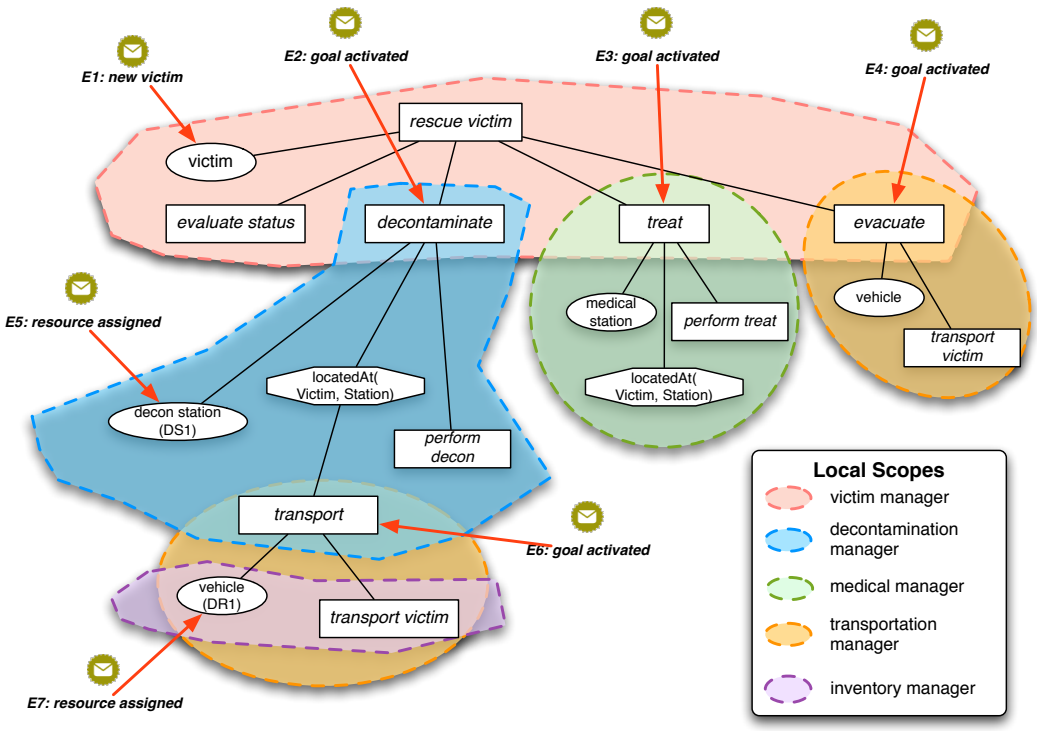


Figure 8.22. Local scope-based event notification in Episode 1

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.

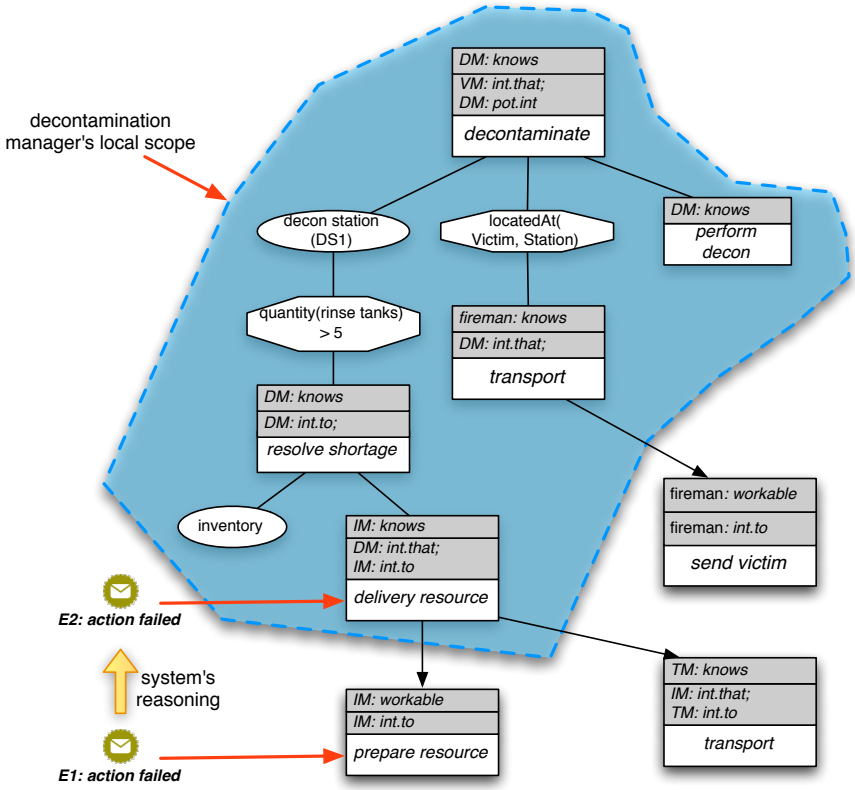


Figure 8.23. The DM's local scope in Episode 2

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 awareness need 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 8.3.1.3, the actors' local scopes change a lot in the development of the collaborative activity. Figure 8.24 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.

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

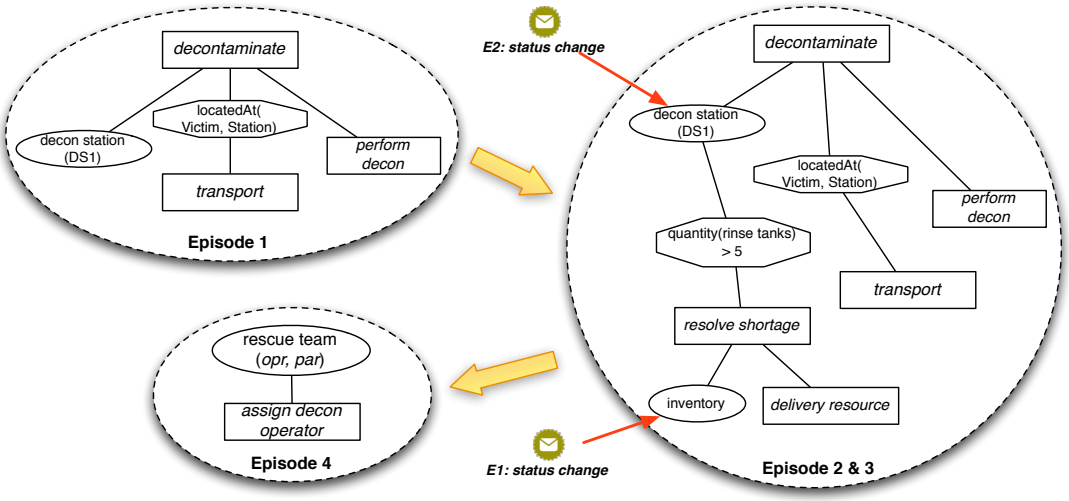


Figure 8.24. The changing local scope of the DM

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.

8.5.2 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 actor's 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 8.25 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.

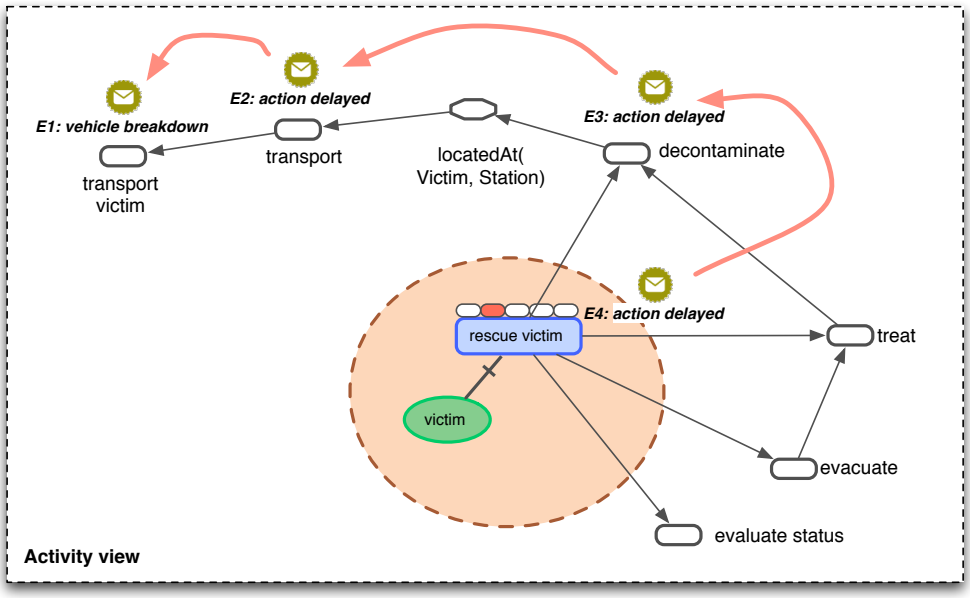


Figure 8.25. Backward tracking in the interpretation of $E4$ in Episode 3

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 8.4.2, 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 8.26 shows the sequence of the events that the decontamination manager needs to look into during the interpretation of $E1$ in Episode 2.

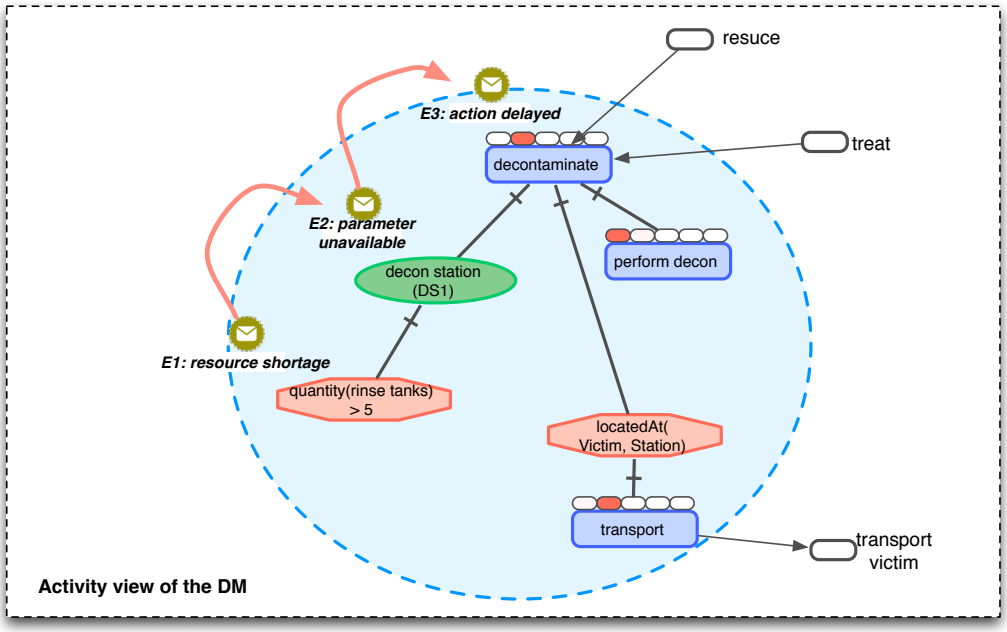


Figure 8.26. Forward tracking in the interpretation of $E1$ in Episode 2

8.5.3 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). The actor externalizes the interpretation as new events, which are then received by other actors. These actors

build their interpretation 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.

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 other's 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 8.27 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 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 Section 3.2.2 clearly shows 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 8.28 illustrates the two cases of event propagation

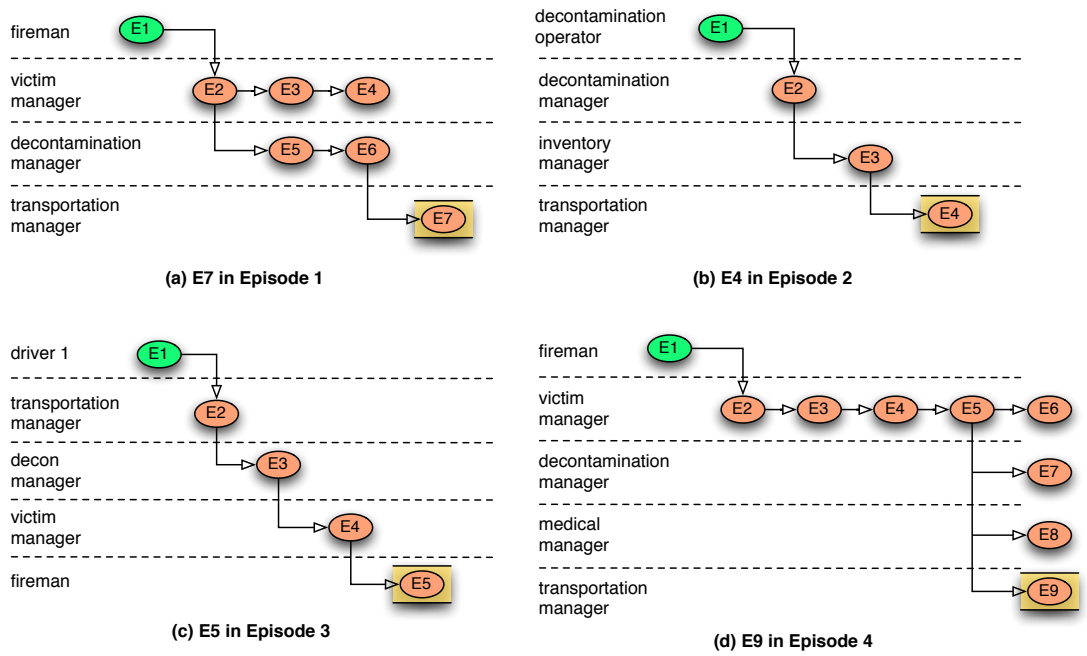


Figure 8.27. Active event propagation chains in the episodes

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, 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

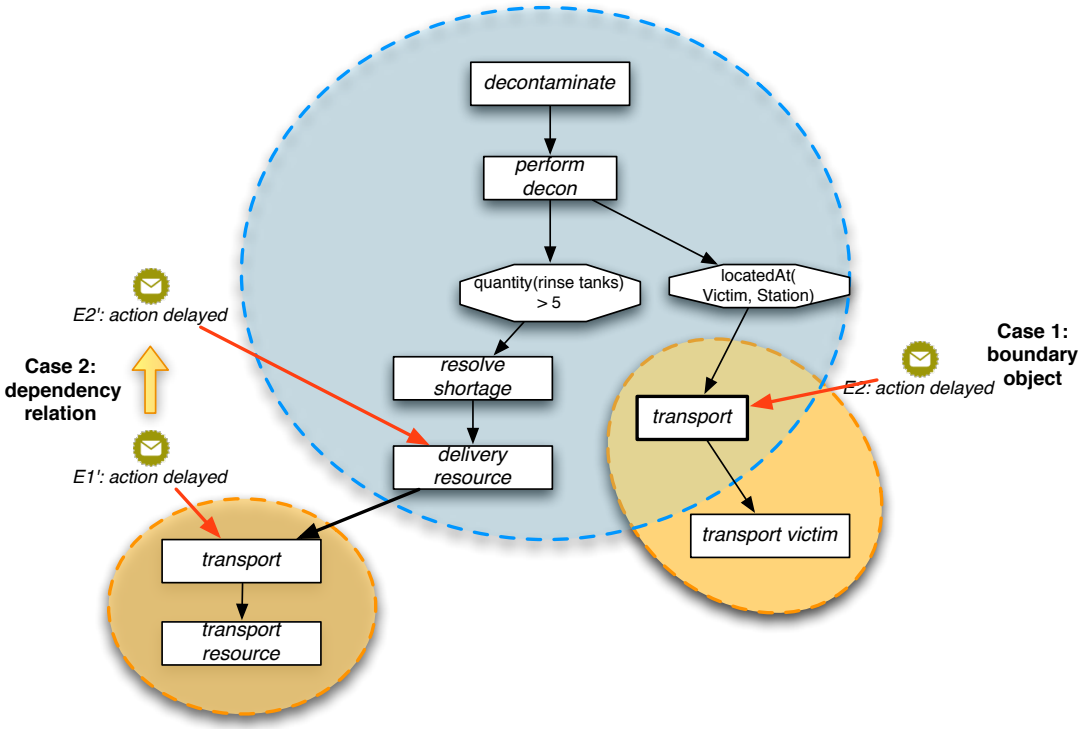


Figure 8.28. Event propagation through boundary objects and dependency relations

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.

8.6 Discussion

This chapter describes a case study in the domain of emergency response to justify our awareness promotion approach.

In Section 8.1, 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.

Section 8.2 shows a concrete scenario of the large-scale distributed collaborative activity in emergency response. The collaborative activity in this study includes a large

number of team workers that engage in a variety of distributed, yet interdependent actions. The four episodes then demonstrate the different types of dynamics that may occur in the scenario, and how the collaborative activity is under continuous development in order to handle the dynamics.

Following the problem scenario, we conduct an analysis of the awareness phenomena in the scenario based on our conceptual model in Chapter 3. The analysis includes a cognitive task analysis and an interaction analysis of the four episodes in the scenario. The former shows the capability of using the conceptual model to understand the major characteristics of collaborative awareness in the scenario, and the latter demonstrates the different developmental trajectories of awareness in the episodes.

To evaluate the expressiveness of the PlanGraph model to represent collaborative activities, and the validity of the knowledge updating processes, we use the EDAP platform to simulate the event-driven development in the four episodes (Section 8.4). The development of the PlanGraph model in the four episodes shows the capability of the system to handle the four typical developmental trajectories of collaborative activities as represented by the four episodes respectively.

In Section 8.5, 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 actor's 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 system's 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 Research contributions

Focusing on the design of a computational system to support awareness in large-scale distributed collaborative activities, this research fits into the design-science paradigm in information science [56]. As argued by Hevner et al [56], effective design-science research must provide clear contributions in at least one of the following areas: (1) the development of constructs or models that extend and improve the understanding of the design problem (i.e. foundations), (2) the development of methods or tools that enable solutions to the design problem (i.e. design artifacts), (3) and the development or creative use of evaluation methods or new evaluation metrics (i.e. methodologies). In this study, we claim contributions in the first two areas.

The conceptual model of awareness in large-scale distributed collaboration The first contribution of this research is the integrated conceptual model for understanding the awareness phenomena in large-scale distributed collaboration. Our conceptual model is built on top of existing theories and models for understanding the awareness phenomena in the literature, and in turn contributes to existing knowledge foundations in two aspects:

1. We adopt several interrelated constructs, i.e. activity, local scope, and dependency, to understand the *product* of awareness phenomena, i.e. what part of the collaborative activity the actors should be aware of. These constructs together enrich the existing understanding of the awareness phenomena in collaborative environments. Beyond the knowledge sharing perspective, we emphasize the distributed nature

of the awareness phenomena. Because of the differences in local scopes, each team member's awareness is partial, but at the same time is compatible for the team to perform collaborative activities successfully.

2. We build on top of these constructs to understand the awareness *process* in collaborative environments. Our model is able to account for how the awareness is distributed across multiple team members. Comparing with existing models, our model provides a better explanation of how the compatibility of different actors' awareness is achieved through the integration of multiple individual cognitive processes and social processes.

Beyond the theoretical contribution, this conceptual model has also shown its value in guiding the design of awareness supporting tools in our study:

1. First, it helps us to identify key design issues to support awareness in large-scale distributed collaborative activities. By conceptualizing the awareness phenomena in distributed, complex collaboration as continuous developed through a variety of cognitive and social processes, the design issues for awareness support can be organized at both 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 team processes in which team members interact with each other to achieve compatible awareness.

2. Second, the conceptual model also guides our design of the computational awareness promotion approach. On one hand, the knowledge representation of our approach is designed to comply with the three constructs in the conceptual framework. As these constructs describe the configuration of the collaborative activity that regulates how the awareness is distributed across team members, they provide the necessary knowledge for the system to reason about human actors' awareness needs and hence promote awareness. Furthermore, one of the design principles of our awareness promotion is to consider the awareness support from a collective perspective and provide support for the developmental trajectories of collaborative awareness, which is motivated by the understanding of the awareness processes in the conceptual framework.

The computational awareness promotion approach The second major contribution of this study is the computational awareness promotion approach. Comparing with

existing awareness support methods, the awareness promotion approach emphasizes the active role of the computer to mediate the awareness processes. While human actors still need to undergo the cognitive processes to develop individual awareness, and use it to make decision and perform actions in their own local scopes; the computer system takes the responsibility to maintain a collective picture of the whole collaborative activity, and utilizes this knowledge to facilitate the various cognitive and social awareness processes among human actors.

Following these design principles, the awareness promotion approach is built on top of two major components: a computational representation of the field of work based on the PlanGraph model, 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.

The awareness promotion approach has several advantages to handle the increased level of complexity and dynamics in large-scale distributed collaborative activities.

1. First, the awareness promotion approach utilizes the computational knowledge representation to model the collaborative activities and offloads some of the representation and reasoning efforts from the human to the computer. Hence, it can handle more complex collaborative configurations than existing awareness models.

2. Meanwhile, the knowledge representation is dynamically updated to reflect the current state of the collaborative work, which allows it to handle increased level of dynamics.

3. Last, as the computer's knowledge representation is at the team level and is equipped with the computational reasoning capabilities, it allows the computer to provide support on both the higher level of individual awareness processes and awareness transactions.

9.2 Future directions

This study has taken a step towards addressing the challenges of supporting awareness in large-scale distributed collaborative activities. However, due to the higher level of complexity and dynamics in these activities, awareness support becomes a much more

difficult task and this study by no means can address all the issues. In the following of this section, we discuss the major limitations and future directions of this study.

Behavioral study to justify the conceptual model As this research follows the design-science paradigm [56], the conceptual model of awareness as we presented in this study is mainly used to improve the understanding of the design problem and identify the design issues. Even though the model is built on top of existing theories and models for understanding the awareness phenomena in the literature, it has not been rigorously tested. Some of the design assumptions in the model, such as the coupling between awareness requirements and the activities, the construction of local scopes from intentions and capabilities, need to be justified in behavioral studies. As argued by Hevner et al. [56], the design science and behavioral science studies should complement each other in information system research. On one hand, the behavioral study seeks to develop and justify theories and models that explain or predict human phenomena surrounding the design activities. On the other hand, the results of design science research provide new artifacts that enable the behavioral study to learn its practical implications. As a result, we believe that the computational approach in this study provides the platform to conduct future behavioral studies to understand the awareness phenomena in large-scale distributed collaboration, which in turn will inform the future iterations of the design.

Usability study of the computational approach Although we have applied several design evaluation methods in the case study to demonstrate the effectiveness of the awareness promotion approach. The evaluation is largely formative and focuses on the utility of the system. However, if we want to know how well the computational approach works in real situations, a usability study becomes relevant. Even though it is a challenging task to evaluate the usability of the whole system, we can design several smaller studies to test the usability of individual components, such as the usability of the visualization of event propagation view, or the direct manipulation interface for event externalization.

Extension to asynchronous collaboration This study has been focused on supporting synchronous collaboration where the awareness events are notified to the actors whenever they occur. The awareness support becomes more interesting when the large-scale collaboration is also asynchronous with a significant time span. In the asynchronous situations, it becomes even more important to maintain the developmental trajectories of the events as people are much easier to lose track of how the events are developed.

Furthermore, in the collaborative activities with a longer developmental history, the higher level of the activity awareness, such as communities of practice, social capital, and human development, becomes more important and needs to be well supported [22].

Extension to multi-tasking The knowledge representation based on PlanGraph model in this study focuses on modeling the collaborative activity with a single shared goal across team members. However, in real world complex collaboration, it is possible that one actor participates in multiple collaborative activities at the same time. The PlanGraph model can be extended to support the multi-tasking situations, where multiple collaborative activities can be modeled at the same time. The multi-tasking situations provide some interesting problems to the knowledge representation, such as how the local scopes of the work should be defined when multiple activities are active, how the dependencies may be posed on two actions of the same actor, but belong to two different collaborative activities.

Appendix A

Events 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: Role 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] M. Benerecetti, P. Bouquet, and C. Ghidini. Contextual reasoning distilled. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(3):279–305, July 2000.

[9] Steve Benford and Lennart Fahlén. A spatial model of interaction in large virtual environments. In *Proceedings of the third conference on European Conference on Computer-Supported Cooperative Work*, pages 109–124, September 1993.

[10] Steve Benford, Chris Greenhalgh, Tom Rodden, and James Pycock. Collaborative virtual environments. *Communications of the ACM*, 44(7):79–85, July 2001.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Susanne Bodker. Computers in Mediated Human Activity. *Mind, Culture, and Activity*, 4(3):149–158, July 1997.

[16] 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.

[17] Guoray Cai, Alan MacEachren, Isaac Brewer, Mike McNeese, Rajeev Sharma, and Sven Fuhrmann. Map-Mediated GeoCollaborative Crisis Management. In Paul Kantor, Gheorghe Muresan, Fred Roberts, Daniel Zeng, Fei-Yue Wang, Hsinchun Chen, and Ralph Merkle, editors, *Proceedings of the IEEE International Conference on Intelligence and Security Informationatics*, volume 3495 of *Lecture Notes in Computer Science*, pages 65–76. Springer Berlin / Heidelberg, 2005.

[18] Guoray Cai, Alan M. MacEachren, Rajeev Sharma, Isaac Brewer, Sven Fuhrmann, and Mike McNeese. Enabling GeoCollaborative crisis management through advanced geoinformation technologies. In *Proceedings of the 2005 national conference on Digital government research*, pages 227–228, May 2005.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] John M. Carroll, Mary Beth Rosson, Umer Farooq, and Lu Xiao. Beyond being aware. *Information and Organization*, 19(3):162–185, July 2009.

[24] 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.

[25] A Dix. Challenges for cooperative work on the web: An analytical approach. *Computer Supported Cooperative Work (CSCW)*, 6(2):135–156, 1997.

[26] 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.

[27] 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.

[28] M J Egenhofer. Deriving the composition of binary topological relations. *Journal of Visual Languages and Computing*, 5(2):133–149, 1994.

[29] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, January 1991.

[30] 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.

[31] M R Endsley. Direct measurement of situation awareness: Validity and use of SAGAT. *Situation awareness analysis and measurement*, pages 147–173, 2000.

[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] E B Entin and E E Entin. Assessing team situation awareness in simulated military missions. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 44, pages 73–76. SAGE Publications, 2000.

[35] 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.

[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. In *Proceedings of the fourth conference on European Conference on Computer-Supported Cooperative Work*, 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] C Ghidini and F Giunchiglia. Local models semantics, or contextual reasoning= locality+ compatibility. *Artificial intelligence*, 127(2):221–259, 2001.

[46] 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.

[47] B J Grosz and S Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[48] Barbara J. Grosz and Luke Hunsberger. The dynamics of intention in collaborative activity. *Cognitive Systems Research*, 7(2-3):259–272, June 2006.

[49] Jonathan Grudin. Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, 37(1):92–105, January 1994.

[50] 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.

[51] 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.

[52] 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.

[53] Mary Hegarty. The Cognitive Science of Visual-Spatial Displays: Implications for Design. *Topics in Cognitive Science*, 3(3):446–474, July 2011.

[54] 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.

[55] D Hernandez, E Clementini, and P Di Felice. Qualitative distances. *Spatial Information Theory A Theoretical Basis for GIS*, pages 45–57, 1995.

[56] 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.

[57] 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.

[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] Scaife M. and Rogers Y. External cognition: how do graphical representations work? *International Journal of Human-Computer Studies*, 45(2):185–213, 1996.

[63] T W Malone and K Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, 26(1):87–119, 1994.

[64] P Markopoulos. A design framework for awareness systems. *Awareness Systems*, pages 49–72, 2009.

[65] J E Mathieu, T S Heffner, G F Goodwin, E Salas, and J A Cannon-Bowers. The influence of shared mental models on team process and performance. *Journal of Applied Psychology*, 85(2):273, 2000.

[66] 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.

[67] D Mendonça and F Fiedrich. Design for improvisation in computer-based emergency response systems. *Proceedings of the 1th International ISCRAM Conference*, 2004.

[68] Gero Mhl, Ludger Fiege, and Peter Pietzuch. Distributed Event-Based Systems. November 2010.

[69] 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.

[70] B A Nardi. *Context and consciousness: activity theory and human-computer interaction*. The MIT Press, 1996.

[71] U Neisser. *Cognition and reality: Principles and implications of cognitive psychology.* WH Freeman/Times Books/Henry Holt & Co, 1976.

[72] A A Nofi. Defining and measuring shared situational awareness. Technical report, DTIC Document, 2000.

[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] U Rauschenbach. Supporting awareness in shared workspaces using relevance-dependent event notifications. *Proceedings of the CVE*, 1996.

[81] Markus Rittenbruch. Atmosphere: A Framework for Contextual Awareness. *International Journal of Human-Computer Interaction*, 14(2):159–180, June 2002.

[82] Markus Rittenbruch and Gregor McEwan. An Historical Reflection of Awareness in Collaboration. *Awareness Systems*, pages 3–48, 2009.

[83] Markus Rittenbruch, Stephen Viller, and Tim Mansfield. Announcing Activity: Design and Evaluation of an Intentionally Enriched Awareness Service. *Human-Computer Interaction*, 22(1-2):137–171, 2007.

[84] 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.

[85] 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.

[86] M B Rosson and J M Carroll. *Usability engineering: scenario-based development of human-computer interaction*. Morgan Kaufmann, 2001.

[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. In *Proceedings of the fifth conference on European Conference on Computer-Supported Cooperative Work*, 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] 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.

[95] 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.

[96] Y Shu and K Furuta. An inference method of team situation awareness based on mutual awareness. *Cognition, Technology & Work*, 7(4):272–287, 2005.

[97] 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.

[98] 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.

[99] 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.

[100] M Sohlenkamp, W Prinz, and L Fuchs. PoliawaC: design and evaluation of an awareness-enhanced groupware client. *AI & Society*, 14(1):31–47, 2000.

[101] MD Spiteri. *An architecture for the notification, storage and retrieval of events.* PhD thesis, University of Cambridge, 2000.

[102] 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.

[103] 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.

[104] A L Strauss. *Continual permutations of action.* Aldine de Gruyter, 1993.

[105] L A Suchman. *Plans and situated actions: the problem of human-machine communication.* Cambridge university press, 1987.

[106] 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.

[107] J.J. Thomas and K.A. Cook. A visual analytics agenda. *IEEE Computer Graphics and Applications*, 26(1):10–13, January 2006.

[108] 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.

[109] W Treurniet, K van BuulBesseling, and J Wolbers. Collaboration awarenessa necessity in crisis response coordination. In *9th International ISCRAM Conference*, Vancouver, Canada, 2012.

[110] 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.

[111] 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.

[112] J Uhlarik and D A Comerford. A review of situation awareness literature relevant to pilot surveillance functions. Technical report, DTIC Document, 2002.

[113] D M Wegner. Transactive memory: A contemporary analysis of the group mind. *Theories of group behavior*, 185:208, 1987.

[114] 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.

[115] 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.

[116] 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.

[117] Bo Yu and Guoray Cai. Coordination of Emergency Response Operations via the Event-Based Awareness Mechanism. In *Proceedings of the 9th International ISCRAM Conference*, Vancouver, Canada, 2012.

[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

Bo Yu received the B.S. degree of Geographic Information Systems (GIS) in 2003 and the M.S. degree of Cartography and GIS in 2006, from School of Earth and Space Sciences, Peking University, China. Since August 2006, he has been a Ph.D. student in the College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA. His research interests include computer supported cooperative work, geographical information retrieval, and geographical information systems.