Int J Software Informatics, Volume 6, Issue 2 (2012), pp. 307–325
International Journal of Software and Informatics, ISSN 1673-7288

E-mail: ijsi@iscas.ac.cn
http://www.ijsi.org
Tel: +86-10-62661040

Towards True Dynamic Workflow for Emergency Response

William Tepfenhart and Jiacun Wang

(Department of Computer Science and Software Engineering, Monmouth University,
West Long Branch, NJ 07728, USA)

Abstract Emergency management is a process by which all individuals, groups, and communities manage hazards in an effort to avoid or ameliorate the impact of disasters resulting from the hazards. Emergency response workflow is dynamic because there are lots of uncertainties with the course of hazard development and rescue effort. Existing dynamic workflow modeling technologies are not sufficient to describe the complex emergency response processes which are context aware and data-driven. In this paper, we propose an intelligent agent based approach to supporting the emergency response process management. The approach integrates BDI (Belief-Desire-Intention) agents with WIFA workflow model, which was developed in our previous work, to a powerful tool for truly dynamic workflow modeling and enactment. A BDI agent is an intelligent agent. Beliefs represent the informational state of the agent - in other words its beliefs about the world. Desires (or goals) represent the motivational state of the agent. They represent objectives or situations that the agent would like to accomplish or bring about. Intentions represent the deliberative state of the agent: what the agent has chosen to do. Intentions are desires to which the agent has to some extent committed. Workflows represent sequences of actions that an agent can perform to achieve one or more of its intentions. Based on this approach, we developed an emergency response training tool which is customizable for individual organization use and scalable to incident response settings from rural to urban domestically and foreign outposts for military applications, and can operate at a holistic exercise level.

Key words: intelligent agent; BDI framework; emergency response systems; dynamical workflow

Tepfenhart W, Wang J. Towards true dynamic workflow for emergency response. *Int J Software Informatics*, **Vol.6, No.2 (2012): 307–325. http://www.ijsi.org/1673-7288/6/i122.htm**

1 Introduction

Emergency management is a process by which all individuals, groups, and communities manage hazards in an effort to avoid or ameliorate the impact of disasters resulting from the hazards. It involves four phases: mitigation, preparedness, response, and recovery. Mitigation efforts attempt to prevent hazards from developing into disasters altogether, or to reduce the effects of disasters when they occur. In

This work is sponsored by the contract W911SR-08-C-0083 through the US Army Research, Development and Engineering Command of the Department of Defense.
Corresponding author: William Tepfenhart, Email: btepfenh@monmouth.edu
Received 2011-03-14; Revised 2011-07-10; Accepted 2011-11-02; Published online 2012-03-19.

the preparedness phase, emergency managers develop plans of action for when the disaster strikes and analyze and manage required resources. The response phase executes the action plans, which includes the mobilization of the necessary emergency services and dispatch of first responders and other material resources in the disaster area. The aim of the recovery phase is to restore the affected area to its previous state. Effective emergency management relies on thorough integration of emergency plans at all levels of government and non-government involvement.

Given the complexity of emergency incidents and emergency response process, it is highly desirable to have a computerized tool to assist emergency managers and first responders in taking appropriate actions in accordance with the departmental plan effectively. A well designed and developed workflow tool could provide process control of the emergency procedures to ensure that they are completed in the correct order and on time. The workload, stress and chance of human error during emergency operation may be reduced with such a tool in place[13].

One of the most important features that emergency response workflows possess is high flexibility[12]. During the course of an emergency event, there are lots of uncertainties which cannot be predicted before it happens. No incident and the response to the incident would follow any predefined fixed workflow to progress. This is totally different from normal manufacturing system workflows where, once the workflow is established, the users just execute it repeatedly without the necessity of frequent modification. Therefore, emergency response workflows are typical of dynamic workflows.

On the other hand, all responding agencies manage people, equipment, facilities, and supplies to accomplish their tasks. However, emergencies can require more specialized and larger quantity of resources than the responding agencies have available. Resource management is a critical part of emergency management and ranges from determining needs to finding and staging resources to meet these needs. Resource Management does not fall under a centralized control element, but is coordinated from the Emergency Operations Centers (EOC) during emergency operations[14]. When an agency receives a resource request, it checks its resource availability. If the agency cannot satisfy the requestor's needs, then the requestor needs to send the request to other agencies. Therefore, the workflow is data driven. The biggest issue here is the coordination between the resource requesting and serving units. Spatial and organizational distribution of the participating emergency management agencies results in distributed knowledge, distributed control and hence suboptimal resource usage. This results in very complex processes with many alternative paths and sections that cannot be planned in advance.

Current emergency response practice is predominantly based on static pre-planning with assumptions being made about the event. This approach suffers inflexibility in the face of novel events characterised by high urgency and the evolving operational conditions[3]. It is impractical to assemble an exhaustive list of potential major events and develop the corresponding response plans. One solution to this is dynamically composing response workflow on-the-fly based on what's happening in the field.

Intelligent Agents have traditionally been applied to the control and optimization of industrial transport and production processes. In contrary to that, research on workflow management and agents in business contexts is more involved with human

processes, typically in the domain of administration and services[2]. In this paper, we introduce the design of an intelligent agent based emergency response workflow model. BDIw (Belief-Desire-Intention-Workflow) agents are used to simulate various types of emergency responding agencies, while a trainee only needs to interact with these simulated agencies in the process of emergency response. A BDIw agent is an intelligent agent. Beliefs represent the informational state of the agent - in other words its beliefs about the world. Desires (or goals) represent the motivational state of the agent. They represent objectives or situations that the agent would like to accomplish or bring about. Intentions represent the deliberative state of the agent: what the agent has chosen to do. Intentions are desires to which the agent has to some extent committed. Workflows represent sequences of actions that an agent can perform to achieve one or more of its intentions. Messages exchanged between agents and the responders are coded in the standard XML format. The intelligent based workflow framework helps achieve a truly dynamic workflow modeling and execution by making on-the-fly decisions on the paths to proceed with based on real-time data and events.

The BDI framework offers a few distinguished advantages which help address the challenges that we have mentioned above. For example, the BDI paradigm is based on folk psychology, where the core concepts of the paradigm map easily to the language people use to describe their reasoning and actions in everyday life [9]. Besides, the BDI paradigm is a relatively mature framework and has been successfully used in a number of medium to large scale software systems. In Ref. [11], Shendarkar et al. applied an extended BDI framework to simulate crowd evacuation from an area under a terrorist bomb attack. As part of on-going research into optimizing the response to large-scale emergencies, an agent-based simulation system developed to evaluate different rescue plans is presented in Ref. [5]. In Ref. [8], agent-based flood evacuation simulation of life-threatening conditions using vitae system model is discussed. The authors of Ref. [3] introduced an integrated framework aimed at adaptive co-ordination of emergency response to dynamic, fast evolving and novel events on a large-scale. The proposed framework consists of a decision-support system and an agent-based simulation of emergency response to large-scale events occurring in real geographical locations. In Ref. [4], Gonzalez presented a crisis response simulation model architecture combining a discrete-event simulation environment for a crisis scenario with an agent-based model of the response organization. There are also other efforts in intelligent agent based emergency response reported, but we didn't find any work that combines the advantage of both workflow technology and software agents in achieving dynamic workflow to emergency response.

Based on the BDIw approach, we developed a training tool which allows EOCs to train their personnel. Traditionally, emergency responders are trained through exercises. There are three major types of exercises: table top exercises, functional exercises and full scales exercises. All these training approaches require significant amount of planning time and are very costly in execution. Therefore, it is of paramount urgent to design and develop a computer-based training tool which allows users to easily construct training scenarios at scales they want to get trained, and conduct the training at the convenience of their desktops. Out tool has many distinguished features, including: (1) It is customizable for individual organization use. (2) It is scalable to

incident response settings from rural to urban domestically and foreign outposts for military applications. (3) It operates at a holistic exercise level.

An appropriate control structures allows the agents to function based on data rather than explicit procedural instructions. With this in mind, our agents are developed such that they are extensible without requiring additional programming. This is to be compared to the JACK BDI programming extension in which all capabilities are explicitly programmed[6]. Effective incident management presents a number of challenges to the responsible agencies[7].

The rest of the paper is organized as follows: BDI agents and WIFA workflow model are briefly introduced in Section II. BDIw framework, its components and its application to emergency response workflow modeling is presented in Section III. An example that shows how the system handles a resource request from an emergency management center step-by-step is given in Section IV. Section V summarizes the contribution of the paper.

2 BDI Agents and WFIA Workflow

The BDIw agent integrates the concepts of a BDI agent and the WIFA workflow model. We briefly describe these two BDIw components in this section.

2.1 BDI agent overview

The belief-desire-intention (BDI) agent architecture is a prominent architecture in agent-oriented software engineering. It is intended for agents that are carrying out "practical reasoning", which covers many real-world applications such as logistics and manufacturing. This is based on the work of the philosopher Michael Bratman[1]. Practical reasoning is defined as reasoning toward actions, as opposed to theoretical reasoning, which is reasoning about beliefs. Practical reasoning can be broken down further into two activities: deliberation (deciding what goals to achieve) and means-end reasoning (how to achieve a goal)[15].

Beliefs represent the informational state of the agent - in other words its beliefs about the world (including itself and other agents). Beliefs can also include inference rules, allowing forward chaining to lead to new beliefs. Typically, this information will be stored in a database (sometimes called a *belief base*), although that is an implementation decision. Desires (or goals) represent the motivational state of the agent. They represent objectives or situations that the agent would like to accomplish or bring about. Examples of desires might be: *find the best price*, *go to the party* or *become rich*. Intentions represent the deliberative state of the agent: what the agent *has chosen* to do. Intentions are desires to which the agent has to some extent committed (in implemented systems, this means the agent has begun executing a plan). Plans are sequences of actions that an agent can perform to achieve one or more of its intentions.

A simple loop of execution for a BDI agent is as follows [10]:

- generate options from event queue;
- deliberate over options;
- update the intentions stack with the selected options;

- execute intentions;
- get new external events;
- drop successful attitudes;
- drop impossible attitudes.

This control flow reflects important components of practical reasoning: option generation, deliberation, execution, and intention handling. But it does not show how an intention is executed and how a plan is structured in general. In this paper, we extend the BDI model by (1) using formal workflows to represent plans to fulfill intentions, and (2) representing each task in a workflow using an adaptor, which is an executable program. The extended BDI model is call BDIw model, where w stands for workflows.

2.2 Basic WIFA workflow model

This section only gives a brief overview of WIFA model. A detailed description is available in Ref. [12].

A workflow is composed of *tasks* that are executed according to some order specified by *precedence constraints*. The *preset* of a task T_k is the set of all tasks that are immediate predecessors of the task, denoted by $^*T_k$; the *postset* of T_k is the set of all tasks that are immediate successors of the tasks, denoted by $T_k{}^*$. If $|T_k{}^*| \geqslant 1$, then the execution of T_k might trigger multiple tasks. Suppose $\{T_i, T_j\} \subseteq T_k{}^*$. There are two possibilities: (1) T_i and T_j can be executed simultaneously, and (2) only one of them can be executed, and the execution of one will disable the other, due to the conflict between them. We denote the former case by $c_{ij} = c_{ji} = 0$, and the latter case by $c_{ij} = c_{ji} = 1$.

In WIFA, a workflow is defined as a 5-tuple: $WF = (T, P, C, A, S_0)$, where

1) $T = \{T_1, T_2, \ldots, T_m\}$ is a set of *tasks*, $m \geqslant 1$.

2) $P = (p_{ij})_{mxm}$ is the *precedence matrix* of the task set. If T_i is the direct predecessor of T_j, then $p_{ij} = 1$; otherwise, $p_{ij} = 0$.

3) $C = (c_{ij})_{mxm}$ is the *conflict matrix* of the task set. $c_{ij} \in \{0, 1\}$ for i= 1, 2, $\ldots$, m and j =1, 2, $\ldots$, m.

4) A= $(A(T_1), A(T_2), \ldots, A(T_m))$ defines *pre-condition set* for each task. $\forall T_k \in T$,$A(T_k)$: $^*T_k \rightarrow 2^{^*T_k}$. Let set $A' \in A(T_k)$. Then $T_i \in A'$ implies $p_{ik} = 1$.

5) $S_0 \in \{0, 1, 2, 3\}^m$ is the *initial state* of the workflow.

A state of a workflow describes the execution status of each task at a time. It is denoted by $S = (S(T_1), S(T_2), \ldots, S(T_m))$, where $S(T_i) \in \{0, 1, 2, 3\}$. $S(T_i) = 0$ means T_i is *not executable* at state S and *not executed previously* (by *previously* we mean before state S is reached); $S(T_i) = 1$ means T_i is *executable* at state S and *not executed previously*; $S(T_i) = 2$ means T_i is *not executable* at state S and *executed previously*; and $S(T_i) = 3$ means T_i is *executable* at state S and *executed previously*.

At the initial state S_0, for any task $T_i \in T$, if there is no T_j such that $p_{ji} = 1$, then $S_0(T_i) = 1$; otherwise $S_0(T_i) = 0$.

A task that has no predecessor does not need to wait for any other task to execute first. In other words, the task is executable immediately. We assume that there is one and only one such task in a workflow, called *start task*. It constitutes the initial trigger

of a workflow. We also assume there is one and one task that has no successors, which is the *end task*. The execution of an end task marks the completion of a workflow.

The dynamics of a WIFA workflow model are captured by state transitions. State transitions are guided by the following two rules:

If $S_a(T_i)S_b$, then $\forall T_j \in T$,

1) If $T_j = T_i$ then $S_b(T_j) = 2$;

2) If $T_j \neq T_i$ then the state value of T_j at new state S_b depends on its state value at state S_a. We consider four cases:

Case A – $S_a(T_j) = 0$:

If $p_{ij} = 1$ and $\exists A \in A\ (T_j)$ such that $S_b(T_k) = 2$ for any $T_k \in A$, then $S_b(T_j) = 1$; otherwise $S_b(T_j) = 0$.

Case B – $S_a(T_j) = 1$

If $c_{ij} = 0$ then $S_b(T_j) = 1$; otherwise $S_b(T_j) = 0$.

Case C – $S_a(T_j) = 2$

If $p_{ij} = 1$ and $\exists A \in A\ (T_j)$ such that $S_b(T_k) = 2$ for any $T_k \in A$, then $S_b(T_j) = 3$; otherwise $S_b(T_j) = 2$.

Case D – $S_a(T_j) = 3$

If $c_{ij} = 0$ then $S_b(T_j) = 3$; otherwise $S_b(T_j) = 2$.

A *well-formed* workflow is a workflow in which there are no dangling tasks and given any reachable state, there is always a path leading the workflow to finish. A confusion-free workflow is a well-formed workflow such that:

1) Either all tasks triggered by the same task are in conflict, or no pair of them is in conflict.

2) A task becomes executable either when all of its predecessor tasks are executed, or when any one of them is executed.

From the perspective of triggering condition and relation among triggered tasks, tasks in a confusion-free workflow can be classified into four types: *AND-in-AND-out*, *AND-in-XOR-out*, *XOR-in-AND-out*, and *XOR-in-XOR-out*. As indicated by the name, for example, a task belongs to this class of *AND-in-AND-Out* iff it is not executable until all its direct predecessor tasks are executed, and after it is executed, all its direct successor tasks can be executed in parallel.

Figure 1 shows a six-task workflow, in which T_3 and T_4 are in conflict (i.e., T_2 is a AND-in-XOR-out task) and T_5 is executable after either T_3 or T_4 is executed (i.e., T_5 is an XOR-in-AND-out task). In WIFA notation, $T = \{T_1, T_2, T_3, T_4, T_5, T_6\}$. P is a 6x6 matrix with $p_{12} = p_{23} = p_{24} = p_{35} = p_{45} = p_{56}$=1 and all other p_{ij}=0. C is also a 6x6 matrix with $c_{34} = c_{43}$=1 and all other c_{ij}=0. $A(T_1) = \varnothing$, $A(T_2) = \{T_1\}$, $A(T_3) = A(T_4) = \{T_2\}$, $A(T_5) = \{\{T_3\}, \{T_4\}\}$, $A(T_6) = \{T_5\}$. The initial state S_0 = (1, 0, 0, 0, 0, 0). T_1 is the only task executable at the initial state S_0. When T_1 is executed, T_2 is triggered, and the new state is S_1 = (2, 1, 0, 0, 0, 0). The execution of T_2 will trigger both T_3 and T_4, and the new state after the execution is S_2 = (2, 2, 1, 1, 0, 0). No we can select either T_3 or T_4 for execution. Suppose we execute T_3 at S_2, then it follows from the state transition rules that the resultant state is S_3 = (2, 2, 2, 0, 1, 0), where $S_3(T_4) = 0$, meaning T_4 is no longer executable because it conflicts with T_3. Executing T_5 at S_3 results in S_4 = (2, 2, 2, 0, 2, 1). Executing T_6 at S_4 results in S_5 = (2, 2, 2, 0, 2, 2), and the workflow execution is finished.

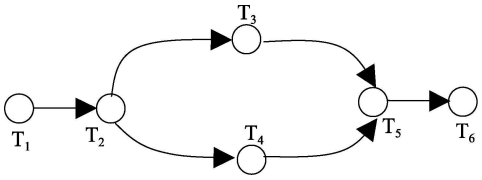

Figure 1. A six-task workflow

2.3 Resource-Constrained workflow model

A resource-constrained workflow is defined as $RCWF = (WF, R^c, R^p, ES_0)$, where

1) *WFCS*: The control flow as defined in Definition 1.

2) $R^c = (R^c(T_1),\ R^c(T_2),\ \ldots,\ R^c(T_m))$ describes the quantity of each type of resource consumption in a task execution, with $R^c(T_k) = \{r^c_{k1}, r^c_{k2}, \ldots, r^c_{kn}\}$, where r^c_{kj} represents the quantity of the resource of type j consumed (or held) when task T_k is executed.

3) $R^p = (R^p(T_1),\ R^p(T_2),\ \ldots,\ R^p(T_m))$ describes the quantity of each type of resource production in a task execution, with $R^p(T_k) = \{r^p_{k1}, r^p_{k2}, \ldots, r^p_{kn}\}$, where r^p_{kj} represents the quantity of the resource of type j produced (or released) when task T_k is executed.

4) $ES_0 = (S_0,\ R_0)$ is the initial state, with $S_0 \in \{0,\ 1,\ 2,\ 3\}^m$ being the state element of the underline basic WIFA workflow WF and R_0, the value of R at the initial state, being the element representing the availability of resources.

Task T_k at state ES_i is said to be executable if and only if it meets all of the following:

1) *Control-Ready*: $S_i(T_k) \in \{1,\ 3\}$, which means T_k is triggered in terms of control flow. In another words, from the control flow aspect of the workflow, the task has to be triggered by its predecessor(s).

2) *Resource-Ready*: $R^c(T_k) \leqslant R_i$, a task, T_k, requires a certain amount of resources in order to be executed. If the current set of resources, R_i, does not have suficient resources required to execute the task, then the task is not executable.

Once the task is executed, the new state $ES_j = (S_j,\ R_j)$ will be determined according to the following rules:

1) S_j is changed according to the state transition rules of basic WIFA workflow.

2) The resource state R_j is derived from the previous resource state R_i, resource consumed by T_k, $R^c(T_k)$, and resource produced by T_k, $R^p(T_k)$, according the following formula:

$$R_j = R_i - R^c(T_k) + R^p(T_k)$$

Notice that the resouce-constrained workflow model presented is slightly different from the one we defined in (Wang, Tepfenhart and D. Rosca 2009), in which decision criteria are part of the model. In BDIw model, the decision-making part of the overall workflow is handled by BDI agents.

3 BDIw Framework

The Belief-Desire-Intention-Workflow (BDIw) Agent model is a variant of the basic BDI model with the introduction of a workflow capability. The basic BDI

model incorporates beliefs, desires, and intentions as the primary sets of information. In this model, the intentions are overloaded with planning and control of the execution of those plans. In the BDIw model, the intentions are associated with plan selection and the workflow is associated with control of the execution of those plans. In this work, additional adaptor programs are used to interact with external data sources.

A BDIw agent can be realized core of five individual components working together along with a suite of adaptor programs that are agent instance specific. The core components are the belief, desire, intention, workflow, and the DBMS components. These core components are data driven and hence will not require modifications to support new agent functionalities. The adaptor programs allow an instantiation of this architecture to communicate and interact with the external sources of data and services. The adaptor components are agent instance specific although a core of basic capabilities can be provided. A basic instantiation of this architecture is shown in Fig. 2.

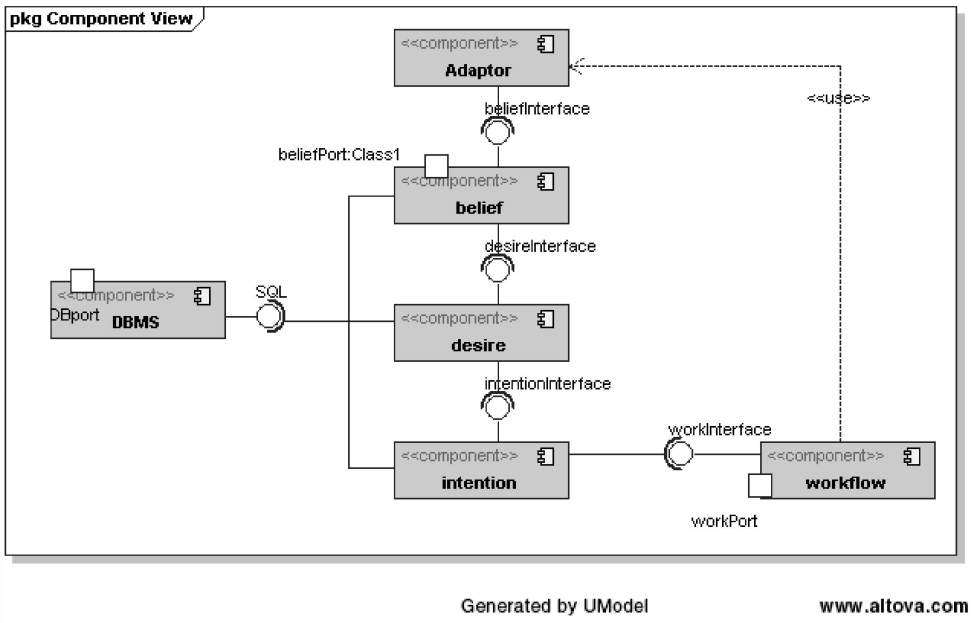

Figure 2. BDIw agent components

This architecture reflects a very definite separation of concerns in that each component provides a very limited and specialized function within the architecture. The development of appropriate control structures allows the agent to function based on data rather than explicit procedural instructions. By maintaining such tight focus, it is possible to create an agent that is extensible without requiring additional programming. This is to be compared to the JACK BDI programming extension in which all capabilities are explicitly programmed.

For examples of the basic WIFA workflow and resource-constrained work, please see Refs. [12,14].

3.1 The brief component

The belief component is responsible for tracking changes in beliefs. All requests to modify the beliefs held by the agent must come through the belief component. There are three major belief changes that it can support: create, delete, and modify. It forwards announcements of such changes to the desire component by identifying the belief that changed and the type of change that was made to it. As shown in Fig. 3, the Brief component interfaces with the Database, Desire and Adaptor components.

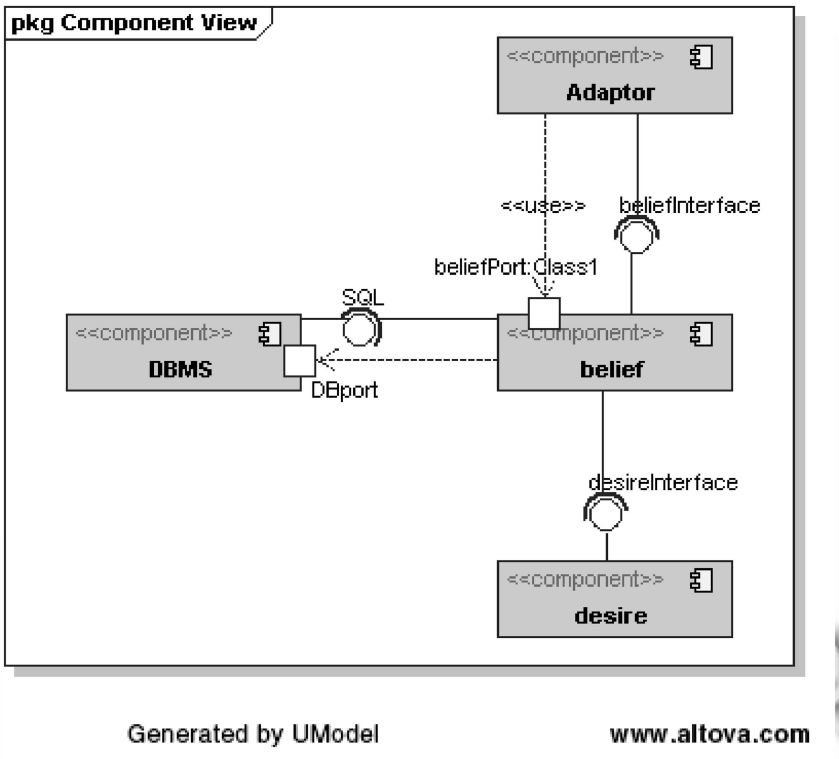

Figure 3. The Brief component

Within the context of the agent architecture, the belief component provides three major capabilities:

1. Services requests from adaptor components
2. Manages the beliefs captured with the database
3. Triggers desires

Upon component startup, the belief component initializes by establishing a connection to the DBMS component and starting up a listener on the input port. During runtime operation, control over the belief component functionality is dictated as follows: When a belief request is acquired, the request is validated for structural and semantic integrity. Then a query is generated based on the request contents and executed with the database. If the request was for a query against known belief then a belief response is created sent to the adaptor, and the component returns to listening

for the next belief request; if the request was for an insert then the database is queried to identify the item id. After that, a desire trigger is generated and sent to the desire component; meanwhile, a belief response is created and sent to the adaptor.

3.2 The desire component

The desire component looks at changes in beliefs along with the complete set of beliefs. It attempts to establish if there is a need to perform some activity to drive what it believes about the world to some state that is more desired. If there is such a need, the desire component announces that need to the intention component by identify the desire (goal) and what triggered that goal. The Desire component interfaces with the Belief, Database, and Intention components, as shown in Fig. 4.

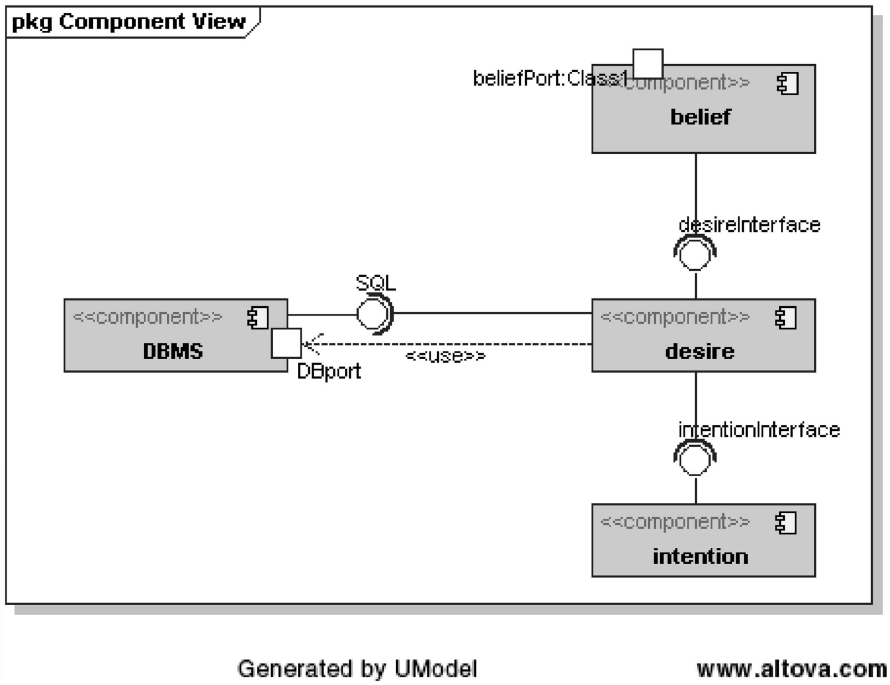

Figure 4. The Desire component

The desire component has two major threads of control: receiving desire triggers and processing desire triggers. The control flow for receiving desire triggers is rather simple. The flow is as follows: First, a desire trigger is received. Then the trigger is acknowledged. Then the trigger is placed in the appropriate stack. Finally any duplicate triggers are removed.

The control flow for processing desire triggers is the more complicated thread of control. The flow starts when the intention component has completed an existing intention or is starting up. First, the external stack is checked for the earliest desire trigger. If the stack contains a desire trigger it is moved over to the internal stack, and the internal stack is checked for desire triggers. If there are desire triggers present then the most recent desire trigger in the internal stack is selected for activation. The desire rule database is checked for all rules that apply and have not been frustrated

previously. If no desire rule is satisfied then the desire trigger is removed from the stack. Otherwise, an intention trigger is constructed and sent to the intention component, and the flow waits for a response to the intention trigger from the intention component. If the intention was not successful then a frustration is inserted into the frustration database.

3.3 The intention component

The intention component, shown in Fig. 5, determines what workflows, if any, can resolve that need given the current state of beliefs. This is done by using the intention to identify all known workflows that support that goal and assessing the resources required for each workflow. It selects the workflow based on availability of resources. Once the intention component identifies a workflow, it invokes the workflow component to execute that specific workflow.

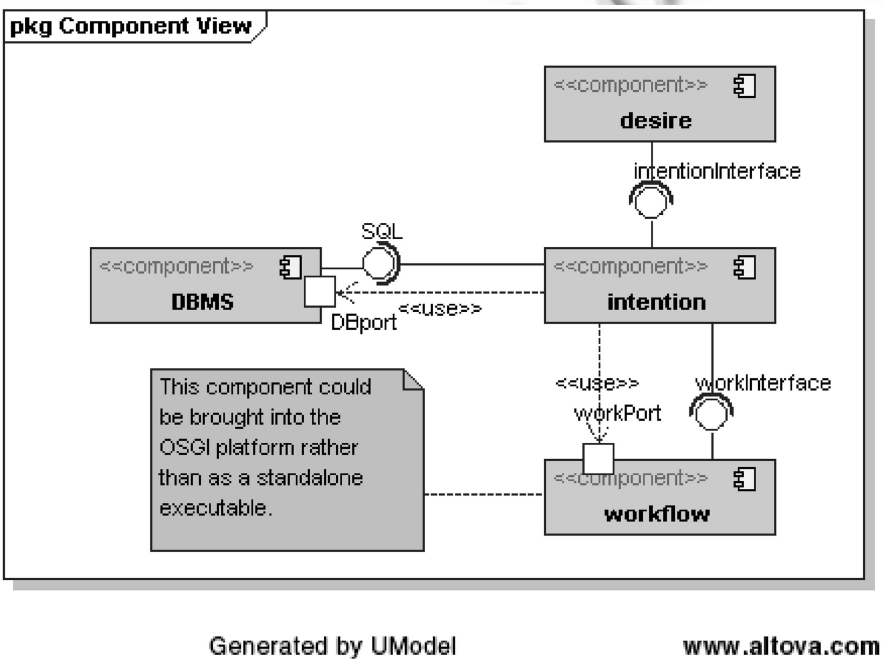

Figure 5. The Intention component

It is possible that multiple workflows satisfy the same desire. A workflow has a priority, which is one criterion that is used to select a workflow among those fulfills the same desire. In addition, each workflow is also associated with resource requirements. When a workflow is selected based on intention and priority, we need to further check if resource requirements are satisfied. A detailed description of the integration of intention components and workflow components is given in the next subsection.

3.4 The workflow component

The workflow component executes the workflow by activating adaptor programs in the manner proscribed by the flow of tasks within the workflow. The main control

process of the workflow component is described in Table 1. This table also shows how the intention component and workflow component are integrated, or in other words, how an entire emergency response workflow is dynamically composed from small individual workflows based on decisions from BDI agents.

Table 1 Main control process of workflow module

```
while (true) {
   receive intention trigger from desire component;
   if intention != NULL                    //intention: intention trigger
push(intention, intention_stack);
   current_intention = pop(intentation_stack);
   if current_intention != NULL {
query(workflow_table, current_intention); if there are returned
entries {
            sort all returned entries in decreasing order of priority;
            current_entry = first entry in the returned entry list;
            while current_entry != NULL
                    if resource query is satisfied {
                    message workflow component to execute the workflow;
                        exit;
                    }
                    else
                    current_entry = next entry in the returned entry list;
                }
                if no workflow selected for lack of resource
                    notify_desire(current_intention, 'insufficient resource');
        }
        else
           notify_desire(current_intention, 'no plan matches the intention');
   }
   else
        sleep for X seconds;    //allow other processes to run
}
```

A task has the following attributes:

```
struct Task {
   name             char[],
   description      char[],
   precedingTasks   Task[],
   succeedingTasks  Task[],
   taskType         {And_In_And_Out,
                    And_in_XOR_Out,
                    XOR_In_And_Out, XOR_in_XOR_Out},
   resource         Resource[]    //Resource is defined below
   application      Filename
} struct Resource {
   name             char[],
   amountRequired   int,
   amountReleased   int
}
```

An external application is hooked up to task through adaptor, which will be discussed in next section. If there is no external application associated with a task, then the value of the `application` field is simply a `NULL`.

We didn't consider timing constraints for tasks in the above definition, because they will make the reasoning of BDIw agents overwhelmingly complicated. It is more realistic to deal only with resource constrained workflows in the first version of implementation of BDIw agents.

Executing a workflow requires resources, which are stored in belief set. There are rules to be enforced regarding resources:

1) When a workflow is selected by the intention component, all resources required for executing this workflow should be reserved only for this workflow. They are not going to be used for other purpose. This avoids potential deadlock due to resource shortages.

2) Since executing a task may consume or release certain resources, the belief set shall be updated each time a task is executed. The resource update may trigger new desire, and the new desire will be inserted to the desire queue of the desire component.

3.5 The adaptor capability

The adaptor programs provide the interface for the agent to interact with external programs and users. It can support peer-to-peer collaborations, user interfaces, or serve as clients within encapsulating client-server enterprise architecture. There can also be instances of adaptor program that execute full time to allow the agent to perform in the role of a server in encapsulating client-server enterprise architecture. An adaptor program can be a very simple single function program that is reusable by multiple workflows.

Depending on the type of adaptor, it will either exit upon successful completion of the task for which it was designed or remain active for use by another task. Some adaptors are active at all times and serve as servers or peers within a larger enterprise architecture. This allows external systems to contact the agent to perform some service. The specific adaptor will dictate the style of interaction (e.g., call-return, asynchronous messaging, etc).

Some examples of adaptor components include:

- User Interface Components
- User Programs
- System Interfaces
- Service Listeners

There is variety even within those basic categories. For example, user interface components could include a message window, a query window, a display list, and an option list that demand a handful of actions from the user. User programs might include opening a word processor, a web browser, or a client program to some external system. System interfaces could support CORBA, SOAP, HTTP, or other protocols by which systems communicate.

3.6 The DBMS capability

The DBMS contains the databases and tables that contain the beliefs along with the data to support the desire and intention capabilities of the BDIw agent. This

component allows the core engine to be tailored to address different domains and support different behaviors.

The DBMS will manage the following tables:

- Belief Tables
- DesireRules
- Frustrations
- IntentionRules
- Plans

The contents of these tables will drive the overall behavior of the BDIw agent. In an environment where multiple agents are active at the same time, it would make sense for each agent to have its own database of beliefs. This is necessary so that individual agents do not enter into conflicts over belief changes. However, each belief can be marked with the agent for which the belief is held. This is a design decision that has not yet been made.

The effect of the different components working together can be a surprisingly complex set of behaviors. The agent will be able to demonstrate opportunistic problem solving in which the latest change in belief can cause the agent to adapt to new situations. It can function in a data-driven and goal-drive manner based upon the types of desires that are managed within it. It can support collaborative interactions with users rather than being limited to master or slave roles with respect to the user.

An agent is extensible in the sense that new belief sets can be added by extending the set of belief tables, new desires added to the desire table, new intentions added to the intentions database, new workflows added to the workflow table, and new adaptor programs made accessible to the workflow engine.

3.7 How it works?

Regardless of how complex it is, an entire dynamic emergency response workflow of an emergency manager is always composed of a set of small and static workflows, each of which represents a basic activity in a rescue effort, such as "request for fire trucks", "set no fly zone", and "report casualties to state EOC". How these small workflows compose to achieve a rescue mission depends on numerous factors with a specific incident and status of the rescue team. In the emergency response word, individuals respond to events. When an event occurs, the individual assesses what need to be done (establishes a desire), selects an action to take to achieve that desire consistent with the resources available and constraints a time, and then executes the procedure. It is well known in the emergency response world that "large plans" (workflows) are likely to fail. Instead, individuals utilizes "small plans" to drive the situation to some better and more manageable state at which point they assess the situation and selects the goal to pursue. These small plans and conditions under which they apply are well known.

We developed an emergency response training tool using this dynamic workflow approach. In this tool, we put all related information regarding local emergency management teams, traffic systems, police department, fire department, hospitals,

and all related resources, etc., to the Belief database. Property damages are also inserted to the Belief table in the runtime. Rules for deciding what goals to take in the granularity of activities are stored in the desire tables, and all workflows which represent the basic activities and their pre-conditions are stored in intention table. This way, when responding to an emergency call, we rely on the intelligent agent to assess the complex situation and suggest the rescue actions in the form of workflows, and responders follow the workflows to execute the rescue plan.

The emergency response training tool was built utilizing agents, as described above, implemented in Java within OSGI environments. Each core agent capability (i.e., belief, desire, intention, control, and task) was implemented as an independent OSGI bundle that provided, on demand, the services dictated by the capability. The database functionality was provided by an MYSQL DBMS. Individual instances of the agent ran within separate OSGI processes to simulate individual responder units (i.e., police cars, firetrucks, HAZMAT teams, EOCs, etc.) with an individual database defined for each responder unit. Communication tasks included components that enabled telephone based conversations, push-to-talk radio conversations, e-mail exchanges, and text messaging with trainees.

4 Example

In this section, we use an example to show how the system responds to an external resource request step by step. This example comes from the emergency response training tool that we developed based on the presented dynamic workflow approach. Each respondent entity has an instance of this agent to simulate its decision making and actions. This example shows a small subset of interaction used in simulating an Office of Emergency Management (OEM). We assume the OEM of township A requests 3 Fire Trucks from the OEM of township B, which is running the proposed BDIw system.

```
Request
  Request Type: ResourceRequest
  Sender: Township A
  Receiver:Township B
  RequestDateTime: 3/1/11 T 12:00:00
  Resource Type: Truck
  Resource Instance: Fire
  Quantity: 3
  DateTimeOut:3/1/11 T 13:00:00
  Destination: Oceanport OEM
```

The Adaptor transforms the above message into an XML message:

```
<beliefRequest>
  <replywith>A12345</replywith>
  <source>External</source>
  <request>
  <table>RESOURCE{\_}REQUEST{\_}TABLE</table>
  <operation>INSERT</operation>
  <field>
    <fieldname>RequestType</fieldname>
      <fieldvalue>ResourceRequest</fieldvalue>
      <fieldtype>String</fieldtype>
    </field>
```

```
<field>
  <fieldname>Sender</fieldname>
  <fieldvalue>Oceanport</fieldvalue>
  <fieldtype>String</fieldtype>
</field>
<field>
  <fieldname>Receiver</fieldname>
  <fieldvalue>Middletown</fieldvalue>
  <fieldtype>String</fieldtype>
</field>
<field>
  <fieldname>Receiver</fieldname>
  <fieldvalue>Middletown</fieldvalue>
  <fieldtype>String</fieldtype>
</field>
<field>
  <fieldname>RequestDateTime</fieldname>
  <fieldvalue>2008-08-08T12:00:00</fieldvalue>
  <fieldtype>DateTime</fieldtype>
</field>
<field>
  <fieldname>ResourceType</fieldname>
  <fieldvalue>Truck</fieldvalue>
  <fieldtype>String</fieldtype>
</field>
<field>
  <fieldname>ResourceInstance</fieldname>
  <fieldvalue>Fire</fieldvalue>
  <fieldtype>String</fieldtype>
</field>
<field>
  <fieldname>Quantity</fieldname>
  <fieldvalue>3</fieldvalue>
  <fieldtype>Number</fieldtype>
</field>
<field>
  <fieldname>DateTimeOut </fieldname>
  <fieldvalue>2008 --08-08T13:00:00</fieldvalue>
  <fieldtype>DateTime</fieldtype>
</field>
<field>
  <fieldname>Destination</fieldname>
  <fieldvalue>OceanportOEM</fieldvalue>
  <fieldtype>String</fieldtype>
</field>
</request>
</beliefrequest>
```

It then sends the message to the Belief Module (BM) which inserts the brief (the request) into table `RESOURCE_REQUEST_TABLE` in the database. If the insertion is successful, the BM invokes the Desire Module (DM) with:

```
Desire.InvokeDesireTrigger(External, Insert,
  RESOURCE{\_}REQUEST{\_}TABLE, 001)
```

The DM will immediately return to the BM an acknowledgement that it has received the trigger. Then the BM generates a belief response that will be sent to the Adaptor. If the insertion is unsuccessful, then an error message will be returned to the BM.

In the successful case, the listener thread of the DM picks up the desire trigger and checks the source of the trigger: whether it is an external or internal generated event. This trigger is an external trigger; therefore the desire goes into the pending queue.

The control thread of the DM is woken up by the desire. It checks the working stack. If there is no trigger there, it checks the pending queue. It finds the trigger there, and moves it to the working stack. It then starts establishing desires for the topmost item in this working stack.

Now the DM selects the desire rules from the DESIRERULES table, in order of their priority. The DM assumes that there is another belief table: RESOURCE, which contains the resources available for the local agent. The DESIRERULES table and RESOURCE table are illustrated in Table 2 and Table 3, respectively.

Table 2 DESIRERULES **table**

Table	Condition	Query	Priority	Desire
RESOURCE _REQUEST _TABLE,	ONE	Count how many fire trucks available in the resource table and compare them with the requester trucks. IF available>requested THEN grant request	3	GRANTREQUEST
RESOURCE _REQUEST, _TABLE,	ONE	IF available $<$ requested and available > 1 THEN partially grant request	3	PARTIALGRANTR EQUEST
RESOURCE _REQUEST _TABLE,	ONE	IF available $= 0$ THEN reject request	3	REJECTREQUEST

The first rule in the DESIRERULES table is selected. The belief is satisfied since it requires only one result to be returned from the query. The frustrated desire table is consulted to assure that this particular desire has not been unsuccessfully attempted for this particular table entry. Since this belief has not been previously frustrated, this thread asserts the desire GRANTREQUEST and sends an intention trigger to the intention module (IM):

```
Intention.receiveTrigger (GRANTREQUEST,
RESOURCE{\_}REQUEST{\_}TABLE,
{001,External)}
```

This thread will wait until a result is returned.

The IM receives the trigger and sorts the applicable workflows according to their priority. Let us assume that the INTENTIONS table is like Table 4.

Table 3 RESOURCE table

ID	ResType	ResInstance	Condition	Location	Availability
001	TRUCK	FIRE	OK	Town B	Y
002	TRUCK	SNOW	OK	Town B	Y
003	TRUCK	FIRE	REPAIR	Town B	N
004	TRUCK	FIRE	OK	Town B	Y
005	TRUCK	FIRE	OK	Town B	Y

Table 4 INTENTIONS table

IntentionID	DesireToken	Workflow	Priority	Query
001	GRANTREQUEST	WF001	HIGH	NONE
002	PARTIALGRANTREQUEST	WF002	MEDIUM	NONE
003	REJECTREQUEST	WF003	LOW	NONE
004	NOTIFYREQUESTOR	WF004	HIGH	NONE
005	DELIVERRESOURCE	WF005	HIGH	NONE
006	TRACKRESOURCE	WF006	HIGH	RADIO WITHIN RANGE
007	TRACKRESOURCE	WF007	MDIUM	NONE

The IM finds the WF001 workflow to satisfy this desire. It checks then to see whether the resources necessary for the workflow are available. Since we have not associated any resource with this workflow, the resource requirements are automatically met. The IM generates a request to the workflow module (WM):

```
AssertTrigger(WF001,EXTERNAL,
RESOURCE{\_}REQUEST{\_}TABLE,001)
```

and waits for a response from the WM.

Upon receiving the trigger from the IM, the WM retrieves the stored workflow from the DB and starts the execution of the workflow. Each task in the workflow will have an attached Adaptor. For example, WF001 has three tasks, and therefore invokes 3 adaptors:

- Query user whether to continue the process
- Select resource
- Update RESOURCE table (assumes sending a message to the BM to update the RESOURCE table).

The last task will trigger another desire to notify the requestor, deliver the resource, and track the resources.

Intention IDs 006 and 007 illustrates two approaches for tracking a resource. In Intention 006, the workflow can only be executed if the fire truck is within radio range. If so, the OEM of town B will call the resource directly. Otherwise, the OEM of town B must use a telephone to call the OEM of town A to get the status of the fire truck.

5 Conclusions

Emergency response workflow is distinguished by its intensive flexibility due to uncertainties with the nature of incidents and numerous factors which could deviate

the emergency response from its path. We presented an extended BDIw framework for emergency response workflow management. The intelligent based workflow framework helps achieve a truly dynamic workflow modeling and execution by making on-the-fly decisions on the paths to proceed with based on real-time data and events. The logic behind of this approach is, regardless of how complex a real emergency response process is, it is always composed of a set of small and static workflows, each of which represents a basic activity in a rescue effort. How these small workflows compose to represent a real emergency response process depends on numerous factors such as resource availability, new events in the course of emergence response, policies of command and control, etc, which in BDIw are modeled by BDI agents and associated tables.

References

[1] Bratman ME. Intentions, Plans, and Practical Reason. Cambridge, MA: Harvard University Press, 1987.

[2] Burmeiste, Birgit, Arnold M, Copaciu F, Rimassa G. BDI-Agents for agile goal-oriented business processes. Proc. of 7th International Conference on Autonomous Agents and Multiagent Systems. Estoril, Portugal, 2008. 37–44.

[3] Coates, Graham, Hawe G, Wilson D, Crouch R. Adaptive co-ordinated emergency response to rapidly evolving large-scale unprecedented events (REScUE). The 8th International ISCRAM Conference. Lisbon, Portugal, 2011. 1–5.

[4] Gonzalez, Rafael A. Crisis response simulation combining discrete-event and agent-based modeling. The 6th International ISCRAM Conference. Gothenburg, Sweden, 2009.

[5] Hawe, Glenn, Coates G, Wilson D, Crouch R. Design decisions in the development of an agent-based simulation for large-scale emergency response. The 8th International ISCRAM Conference. Lisbon, Portugal, 2011.

[6] Howden N, Rönnquist R, Hodgson A, Lucas A. JACK intelligent agents: summary of an agent infrastructure. The 5th International Conference on Autonomous Agents. Montreal, Canada, 2001.

[7] Jain S, McLean CR. An integrated framework for modeling and simulation for incident management. Journal of Homeland Security and Emergency Management, 2006, 3(1).

[8] Liu Y, Okada N, Shen D, Li S. Agent-based flood evacuation simulation of lifethreatening. Journal of Natural Disaster Science, 2009, 31(2): 33–41.

[9] Norling E. Folk psychology for human modeling: extending the BDI paradigm. International Conference on Autonomous Agents and Multi Agent System. New York, 2004.

[10] Rao Anand S, Michael P Georgeff. BDI Agents: From Theory to Practice. Technical Report 56, AAII, 1995.

[11] Shendarkar A, Vasudevan K, Lee S, Son Y. Crowd simulation for emergency response using BDI agent based on virual reality. Proc. of the 2006 Winter Simulation Conference. 2006. 546–553.

[12] Wang J, Rosca D, Tepfenhart W, Milewski A, Stoute M. Dynamic workflow modeling and analysis in incident command systems. IEEE Trans. on Systems, Man and Cybernetics, Part A, 2008, 38(5): 1041–1055.

[13] Wang J, Rosca D, Tepfenhart W, Milewski A. Incident command system workflow modeling and analysis: a case study. Third International Conference on Information Systems for Crisis Response and Management. Newark, NJ, 2006.

[14] WWang J, Tepfenhart W, Rosca D. Emergency response workflow resource requirements modeling and analysis. IEEE Trans. on Systems, Man and Cybernetics, Part C, 2009, 39(3).

[15] Wooldridge M.Reasoning about rational agents. MIT Press, 2000.