Incident Command System Workflow Modeling and Analysis: A Case Study

Jiacun Wang, Daniela Rosca, Williams Tepfenhart and Allen Milewski
Department of Software Engineering
Monmouth University
West Long Branch, NJ 07728
{jwang, drosca, wtepfenh, amilewski}@monmouth.edu

ABSTRACT

The dynamics and volunteer-based workforce characteristics of incident command systems have raised significant challenges to workflow management systems. Incident command systems must be able to adapt to ever changing surroundings and tasks during an incident. These changes need to be known by all responsible parties, since people work in shifts, get tired or sick during the management of an incident. In order to create this awareness, job action sheets and forms have been created. We propose a paperless system that can dynamically take care of these aspects, and formally verify the correctness of the workflows. Furthermore, during an incident, the majority of workers are volunteers that vary in their knowledge of computers, or workflows. To address these challenges, we developed an intuitive, yet formal approach to workflow modeling, modification, enactment and validation. In this paper, we show how to apply this approach to address the needs of a typical incident command system workflow.

Keywords

Incident command systems, workflow, modeling, verification

INTRODUCTION

Incident command systems (ICS) resulted from the need for a new approach to the problem of managing rapidly moving wildfires in the early 1970s. At that time, emergency managers faced a number of problems, such as too many people reporting to one supervisor, lack of reliable incident information, and inadequate and incompatible communications. ICS is now widely used throughout the United States by fire agencies, and is increasingly used by law enforcement and many other public safety organizations for emergency and incident management. These different types of ICS share some common characteristics, such as the incidents occur with no advance notice; the incidents develop rapidly; there are often multiple agencies responsible for the incidents; and risk of life and property loss can be high.

Compared with business related workflows, a distinguishing feature of an ICS workflow is the need of considerable flexibility. It should grow or shrink to meet different needs. This makes it a very cost-effective and efficient management system. Although workflow is an old concept (Zisman, 1977) its research and implementation are gaining momentum due to the increasing dynamics and the continuous changes of the market places. The business environment today is undergoing rapid and constant changes. The way companies do business, including the business processes and their underlying business rules, ought to adapt to these changes flexibly with minimum interruption to ongoing operations (Dourish, 2001; Marinescu, 2002). This flexibility becomes of a paramount importance in applications such as an ICS.

Moreover, an ICS would support the activities necessary for the allocation of people, resources and services in the event of a major natural or terrorist incident, and have to deal with frequent changes of the course of actions dictated by incoming events, a predominantly volunteer-based workforce, the need to integrate various software tools and organizations, a highly distributed workflow management.

Dealing with these issues generates many challenges for a workflow management system. The need of making many ad-hoc changes calls for an on-the-fly verification of the correctness of the modified workflow. This cannot be achieved without an underlying formal approach of the workflow, which does not leave any scope for ambiguity and sets the ground for analysis. Yet, since our main users will be volunteers from various backgrounds, with little computer experience, we need to provide a tool with highly intuitive features for the description and modification of the workflows.

A number of formal modeling techniques have been proposed in the past decades. Among them, Petri nets have been widely accepted as a very powerful tool (Aalst, 1996; Rosca et al., 2002; Wang, 1998). Other than Petri Nets, techniques such as state charts have also been proposed for modeling workflow management systems (Lawrence, 1997). Although state charts can model the behavior of workflows, they have to be supplemented with logical specifications for supporting analysis. Singh et al (Singh et al, 1995) used event algebra to model the inter-task dependencies and temporal logic. Attie et al (Attie et al, 1993) used computational tree logic to model tasks by providing their states together with significant events corresponding to the state transitions (start, commit, rollback etc) that may be forcible, rejectable, or delayable.

As indicated in (Aalst et al., 2003), it is desirable that a business process model can be understood by the various stakeholders involved in an as straightforward manner as possible. Unfortunately, a common major drawback that all the above formal approaches suffer is that only users who have the expertise in these particular formal methods can build their workflows and dynamically change the business rules within the workflows. For example, in order to add a new task to a Petri-net based workflow, one must manipulate the model in terms of transitions, places, arcs and tokens, which can be done correctly and efficiently only by a person with a good understanding of Petri-nets. This significantly affects the application of these approaches on a large scale.

To address this issue, we introduced a new formalism for the modeling and analysis of workflows (WIFA – A Workflows Intuitive, yet Formal Approach), which, in addition to the abilities of supporting workflow validation and enactment, possesses the distinguishing feature of allowing users who are not proficient in formal methods to build up and dynamically modify the workflows that address their business needs in (Wang et al, 2005). We developed a set of theorems which help verify the well-formedness of a workflow after a change (Wang and Rosca, 2006). In this paper, after a short introduction of WIFA, we show how to apply WIFA to ICS workflow modeling, modification and validation.

THE WIFA WORKFLOW MODEL

A workflow consists of processes and activities, which are represented by *tasks*. Two tasks are said to have *precedence constraints* if they are constrained to execute in some order. 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 $|T_k{}^*| \geq 1$, 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 $c_{ij} = c_{ji} = 1$.

To increase modeling capability, we adopt the *OR precedence constraint* model, which allows a task to be executed when *some* of its immediate predecessors are executed.

Workflow Definition

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*, $m \geq 1$.
2) $P = (p_{ij})_{m \times m}$ is the *precedence matrix* of the task set. If T_i is the immediate predecessor of T_j, then $p_{ij} = 1$; otherwise, $p_{ij} = 0$.
3) $C = (c_{ij})_{m \times m}$ 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 the *WF* 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*; $S(T_i) = 1$ 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*.

By the above definition of state values, at any state, only those tasks whose values are either 1 or 3 can be selected for execution. Suppose task T_i at state S_a is selected for execution, and the new state resulted from the execution of T_i is S_b, then the execution of T_i is denoted by $S_a(T_i)S_b$.

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 $S_0(T_i) = 0$.

Note that tasks that have no predecessors do not need to wait for any other task to execute first. In other words, these tasks are executable immediately. We assume that there are always such tasks in a workflow. They are the initial triggers or "starting" tasks of workflows.

Fig. 1 shows a workflow model with seven tasks, $T = \{T_1, T_2, \ldots T_7\}$, in which T_1 is the starting task of the workflow. The execution of T_1 triggers both T_2 and T_3, which do not conflict with each other, i.e., $c_{23} = c_{32} = 0$. T_2 can be triggered by either T_1 or T_6, i.e., $A(T_2) = \{\{T_1\}, \{T_6\}$. The execution of T_5 triggers both T_6 and T_7, which conflict with each other, i.e., $c_{67} = c_{76} = 0$. T_7 is executable only if both T_2 or T_5 are executed, i.e., $A(T_7) = \{\{T_3\}, \{T_5\}$. The initial state is $S_0 = (1, 0, 0, 0, 0, 0, 0)$. After the execution of T_1, the new state will be $S_1 = (2, 1, 1, 0, 0, 0, 0)$. If in the next step we select T_2 for execution, the new state will be $S_2 = (2, 2, 1, 1, 0, 0, 0)$.

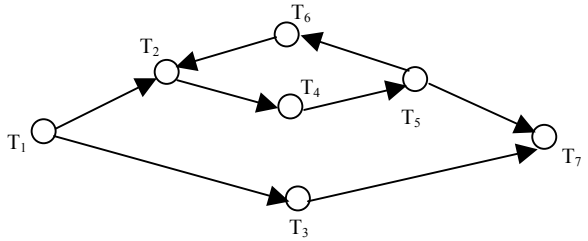

Fig. 1 A seven-task workflow.

State Transition Rules

The dynamics of a WIFA workflow can be captured by state transitions. The state transitions are guided by the following 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 $S_b(T_j) = 2$.

Note that a state value can increment from 0 to 1, from 1 to 2 or from 2 to 3; it can also decrement from 1 to 0 or from 3 to 2. But it cannot decrement from 2 to 1.

Although WIFA was designed with a high degree of usability in mind, it has not sacrificed expressive power. As such, WIFA is able to model sequential and concurrent execution of tasks, conflict resolution, synchronization, mutual exclusion and loops. MILANO (Agostini and DeMichelis, 2000), another tool that claims "simplicity" of use, has sacrificed some expressive power, such as the representation of loops, for the flexibility during enactment. The same deficiency can by noticed in WASA (Weske, 2001). WIDE (Casatti et al, 1998) proposed a complete and minimal set of primitives that allow the correct transformation of an old workflow schema to a new one. Based on that minimal set, other change primitives can be derived, for both modifying workflow schemas and migrating instances to new schemas. TRAM (Kradolfer and Geppert, 1999), uses a versioning approach for the modification of

workflow schemas. They use a principle similar to WASA's for migrating workflow instances to new schemas, e.g. verifying whether the instance can continue from its current state according to the new schema. Currently, WIFA does not handle data flow control, as in WASA, Flow Nets (Ellis and Keddara, 2000), ADEPT (Reichert and Dadam, 1998), and other systems. This represents a dimension that needs to be added to our work.

Well-Formed Workflow Definitions

Two important subclasses of workflows were introduced in WIFA: well-formed workflows and confusion-free workflows. Their formal definitions can be found in (Wang, Rosca, 2005). Here we only give their informal definitions. A workflow is *well-formed* if and only if the following two *behavior conditions* are met:

- There is no dangling task.
- Given any reachable state, there is always a path leading the workflow to finish.

The workflow of Fig. 1 is well-formed, because every task in this workflow is executable, and it is always guaranteed to finish.

To further simplify the workflow structure, we introduce *confusion-free* workflows, which are a class of well-formed workflows. A well-formed workflow is *confusion-free* if and only if the following two *structural conditions* are met:

- Either all tasks triggered by the same task are in conflict, or no pair of them is in conflict.
- 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 well-formed workflow can be classified into four types:

1) *AND-in-AND-out* A task belongs to this class iff it is not executable until all its immediate predecessor tasks are executed, and after it is executed, all its immediate successor tasks can be executed in parallel.
2) *AND-in-XOR-out* A task belongs to this class iff it is not executable until all its immediate predecessor tasks are executed, and after it is executed, only one of its immediate successor tasks can be executed.
3) *XOR-in-AND-out* A task belongs to this class iff it is executable as long as one of its immediate predecessor tasks is executed, and after it is executed, all its immediate successor tasks can be executed in parallel.
4) *XOR-in-AND-out* A task belongs to this class iff it is executable as long as one of its immediate predecessor tasks is executed, and after it is executed, only one of its immediate successor tasks can be executed.

Without loss of generality, a task with only one or no immediate predecessor is treated as an "AND-in" task, and a task with only one or no immediate successor is treated as an "AND-out" task. We denote by T_{AP} the set of all AND-in-AND-out tasks, T_{AC} all AND-in-XOR-out tasks, T_{OP} all XOR-in-AND-out tasks, and T_{OC} all XOR-in-AND-out tasks. For example, for the workflow of Fig. 1, $T_{AP} = \{T_1, T_3, T_4, T_6, T_7\}$, $T_{AC} = \{T_5\}$, $T_{OP} = \{T_2\}$, and $T_{OC} = \varnothing$.

There are two types of loops undesired in a well-formed workflow. One is an infinite loop, where some tasks in the loop execute endlessly. Another type is an inactive loop, where no task can be triggered. An infinite loop has no exit task, while an inactive loop has no entry task. We call these two types of loops *unhealthy loop*s. All other loops are *healthy loops*.

We developed a set of theorems which help verify the well-formedness of a workflow after a change (Wang and Rosca, 2006). Theorem 1 is for the case of adding a new task, Theorem 2 for deleting a precedence constraint, Theorem 3 for deleting a task, and Theorem 4 for adding a constraint. We will apply these theorems to our case study, an ICS incident commander workflow, in the next section. We denote by $\mathcal{R}(WF)$ the set of all reachable states of workflow WF.

Theorem 1: Given a confusion-free well-formed workflow $WF = (T, P, C, A, S_0)$, by adding a new task T_k to it, the obtained new workflow is denoted by $WF' = (T', P', C', A', S_0')$. Then WF' is also a confusion-free workflow if it matches one of the following cases:

1) $^*T_k = T_k^* = \varnothing$, i.e., $p'_{ki} = p'_{ik} = 0$ for all $T_i \in T' \setminus \{T_k\}$.
2) $^*T_k = \varnothing$, $T_k^* \neq \varnothing$, and $\forall T_i \in T_k^*$, if T_k is an AND-in task in WF, then T_i is also an AND-in task in WF'; If T_i is an XOR-in task in WF, then T_i is also an XOR-in task in WF'.

3) $*T_k \neq \varnothing$, $T_k* = \varnothing$. If T_k is an and-in task in WF, then there exists a $S_a \in \mathcal{R}(WF)$ such that $S_a(T_i) = 2$ for all $T_i \in *T_k$; If T_k is an or-in task in WF, then there exists a $S_a \in \mathcal{R}(WF)$ such that $S_a(T_i) = 2$ for some $T_i \in *T_k$. In addition, $\exists T_i \in *T_k$, if T_i triggers two or more conflicting tasks, then T_k conflicts with each of these tasks, otherwise, $c_{kj} = 0$ for any $T_j \in T_i*$.
4) $*T_k \neq \varnothing$, $T_k* \neq \varnothing$, with all other conditions appear in 2) and 3). Besides, $\forall T_i \in T_k*$, if T_i is also a predecessor of T_k (i.e., T_k introduces a loop), then T_i can only be an XOR-in task, and the loop is a healthy loop.

Theorem 2: Let $WF = (T, P, C, A, S_0)$ be a confusion-free well-formed workflow with $T_i, T_j \in T$ and $p_{ij} = 1$. WF' is obtained by deleting the precedence arc between T_i and T_j, i.e. setting p_{ij} to 0. Then WF' is confusion-free well-formed iff the deletion does not introduce any unhealthy loop.

Theorem 3: Given a confusion-free well-formed workflow $WF = (T, P, C, A, S_0)$, deleting a task $T_k \in T$ and all precedence arcs starting from or ending up to T_k, the obtained new workflow is denoted by $WF' = (T', P', C', A', S_0')$. If the deletion does not cause an unhealthy loop, then WF' is also a confusion-free well-formed workflow.

Theorem 4: Let $WF = (T, P, C, A, S_0)$ be a well-formed confusion free workflow with $T_i, T_j \in T$ where $p_{ij} = 0$ and T_i and T_j are not mutual exclusive. WF' is obtained by adding a precedence arc between T_i and T_j, i.e. setting p_{ij} to 1. Then WF' is confusion-free well-formed if it matches one of the following cases:

1) T_i is of the same type in both WF and WF', so is T_j.
2) If the addition introduces a loop, then the loop must be healthy.

INCIDENT COMMAND SYSTEM WORKFLOW MODELING

Every incident or event has certain major management activities or actions that must be performed. Even if the incident is small, these activities will still apply to some degree. As shown in Fig. 2, the ICS organization is built around five major management activities:

- *Command* -- Sets objectives and priorities and has overall responsibility at the incident or event.
- *Operations* – Conducts tactical operations to carry out the plan, develops the tactical objectives, and organizes and directs all resources.
- *Planning* -- Develops the action plan to accomplish the objectives, collects and evaluates information, and maintains resource status.
- *Logistics* -- Provides support to meet incident needs and provides resources and all other services needed to support the incident.
- *Finance/Administration* -- Monitors and analyzes costs related to incident.

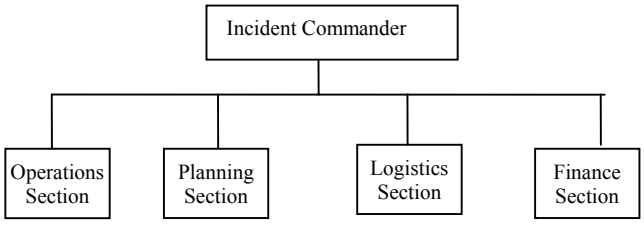

Fig. 2. Structure of a typical ICS

These five major management activities lay at the foundation of every ICS established by the National Incident Management System (NIMS). They apply for handling any type of incident, ranging from a routine emergency, planning for a major event, or managing the response of a major disaster.

On small incidents, these major activities may be managed by one person, the Incident Commander (IC). Large incidents usually require different persons to fulfill the roles shown in Fig. 2.

Incident Commander Workflow Modeling

In an ICS, each section chief has a job action sheet, which documents all tasks the section chief needs to do in case of an incident. Table 1 shows an example job action sheet of an incident commander. There are 23 individual tasks in total, which are grouped into three categories according to their urgencies: immediate, intermediate and extended. The job action sheet only lists all tasks to be executed, but does not tell what are the temporal relationships among the tasks and what is the exact order to execute these tasks. They could be modified during the evolution of an emergency due to incoming events. What happens with the modified workflow? Is it still correct? We believe an ICS could significantly benefit from the formal representation of the tasks, allowing various analyses of the workflows.

Table 1 Example job action sheet of an incident commander

Immediate
- INIT: Initiate the system
- RJAS: Read entire job action sheet.
- PIDV: Put on position identification vest.
- ALSC: Appoint logistics section chief.
- APSC: Appoint planning section chief.
- AFSC: Appoint finance section chief.
- AOSC: Appoint operation section chief.
- ASAM: Announce a status/action plan meeting with all section chiefs.
- RSRL: Receive status report from the logistics section chief.
- RSRP: Receive status report from the planning section chief.
- RSRF: Receive status report from the finance section chief.
- RSRO: Receive status report from the operations section chief.
- DIAP: Discuss an initial action plan with all section chiefs.
- DALS: Determine appropriate level of service during immediate aftermath.
- RIFD: Receive initial facility damage survey report from the logistic section chief, and evaluate the need for evacuation.
- OPCS: Obtain patient census and status from the planning section chief.

Intermediate
- DRB: Designate routine briefings with section chiefs.
- UAP: Update the action plan regarding the continuance and termination of the action plan.
- CNR: Consult with section chiefs on needs for resources.
- ARR: Authorize requested resources.
- SDRB: Stop routine briefing.
- SCNR: Stop consulting.

Extended
- OSVP: Observe all staff, volunteers and patients for signs of stress and inappropriate behavior.
- RCTP: Report concerns to PSUL.
- PRPR: Provide for staff rest periods and relief.
- SOSV: Stop observing.

The WIFA model of this incident commander workflow is shown in Fig. 4. There can be distinguished three types of tasks:

- **AND-in-AND-out**:
  INIT, RJAS, PIDV, ALSC, APSC, AFSC, AOSC, ASAM, RSRL, RSRP, RSRF, RSRO, DIAP, DALS, RIFD, OPCS;
- **AND-in-XOR-out**:
  UAP, ARR, PRPR;
- **XOR-in-AND-out:**
  DRB, CNR, OSVP.

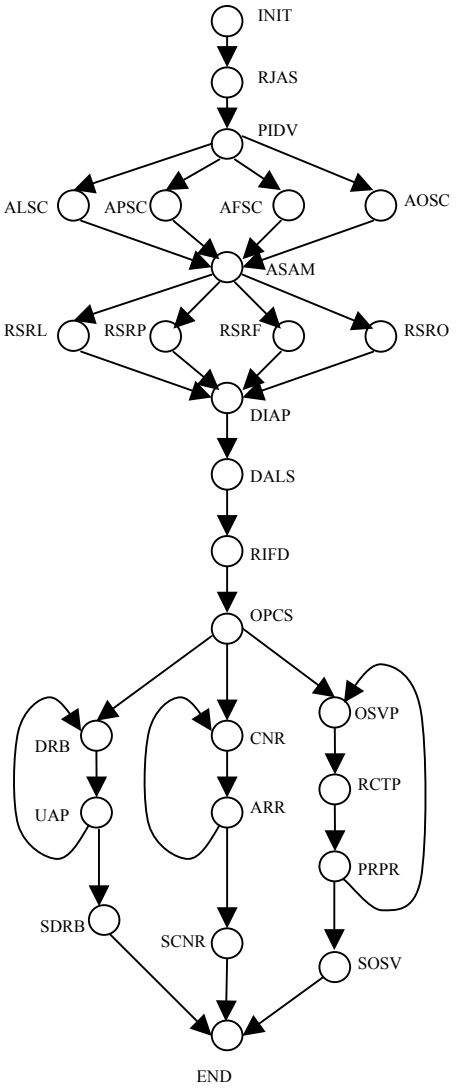

Fig. 3 WIFA model of the incident commander workflow

The model specifies the workflow as follows: After the incident commander initiates the system (task INIT), he will read his job action sheet first (task RJAS), then put on his position identification vest (task PIDV). These three tasks are executed sequentially. In the next step, the commander will appoint the four section chiefs (tasks ALSC, APSC, AFSC and AOSC). Note that the commander can assign the four chiefs in a random order, and the four assignment processes can run simultaneously. So, the predecessor task PIDV is an *AND-in-AND-out* task. But the next task, ASAM, won't start until all the appointments are finished, which means ASAM is an *AND-in* task. After all the section chiefs are appointed, the commander will schedule a meeting with them (task ASAM), and during the meeting, he will receive a status report from each of the section chiefs (tasks RSRL, RSRP, RSRF, and RSRO). There is no particular order on the four status reports, and the reports can be performed simultaneously to the commander and his staff. This means ASAM is also an *AND-out* task. The next task to these four status reports tasks won't be executed until all these four status reports tasks will be finished. So, the task DIAP is an *AND-in* task.

The next three tasks, DALS, RIFD and OPCS, are executed sequentially. After that, there are three parallel task sequences, formed by tasks (DRB, UAP}, {CNR, ARR}, and {OSVP, RCTP, PRPR}, and each of them runs repeatedly until the commander decides to stop them. Then the workflow ends. So, the tasks DRB, CNR, OSVP are all *XOR-in* tasks, UAP, ARR, PRPR all *XOR-out* tasks, and the artificial task END an *AND-in* task.

Enforcement of Well-Formedness

The standard reachability analysis can be used to show that the model of Fig. 3 is well-formed. However, reachability analysis can be very time-consuming even for a moderate sized workflow model. Instead, to enforce the well-formedness of WIFA models we propose a progressive analysis during the construction of the workflow, based on the theorems introduced in the previous section.

We start to build the workflow by introducing the first task, INIT. Of course, the single task workflow is well-formed. Then we add RJAS, the second task, and connect it to INIT. Note that RJAS* = ∅, A(RJAS) = {{INIT}}, i.e., RJAS is an *AND-in* task. Since task INIT is executable, so according to Theorem 1, Case 3), the new workflow of two sequential tasks is well-formed and confusion-free. By the same token, when we add the third task, PIDV, to the workflow, the new one is still well-formed and confusion-free.

Task PIDV has four immediate successors: ALSC, APSC, AFSC and AOSC. When we add the first one of them, say ALSC, to the workflow of tasks INIT, RJAS and PIDV, we can easily prove the obtained workflow composed of INIT, RJAS, PIDV and ALSC is well-formed and confusion-free, based on the same analysis we performed when we added task RJAS. Now let us say we want to add task APSC. As illustrated in Fig. 4, APSC* = ∅, A(APSC) = {{PIDV}}, so APSC is an *AND-in* task, just like RJAS. What makes the addition of APSC different from RJAS is that PIDV, the only immediate predecessor, has multiple immediate successors (PIDV* = {ALSC, APSC}). Since we define PIDV as an "*AND-out*" task, according to Theorem 1, Case 3, the obtained workflow as shown in Fig. 4 is still well-formed and confusion-free. For the same reason, when we add AFSC and AOSC to the workflow, it is still well-formed and confusion-free.

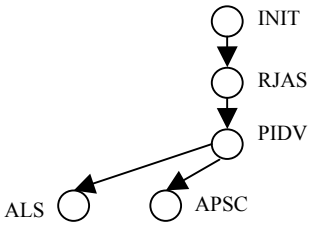

Fig. 4 Add task APSC

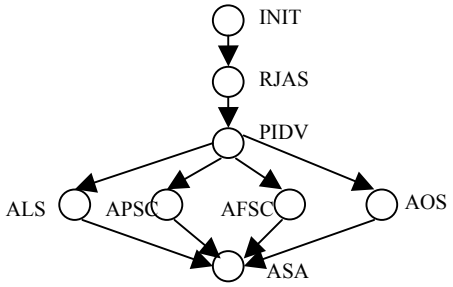

Fig. 5 Add task ASAM

Now, let us consider what happens after we add ASAM to the workflow, as shown in Fig. 5. Because PIDV is executable, and ALSC, APSC, AFSC and AOSC are all its immediate successors, so these four tasks are also executable (this conclusion can also be drawn from the fact that the workflow before adding ASAM is well-formed). Moreover, ALSC, APSC, AFSC and AOSC can be executed in parallel; therefore, there exists a state where all of these four tasks are executed. Examining Theorem 1, Case 3, we know the new workflow is still well-formed and

confusion-free. Repeatedly applying the above analysis will show the workflow including up to task DRB is well-formed and confusion-free.
The next task we are going to add is task UAP. As shown in Fig. 6, UAP introduce a loop into the workflow. Note that DRB is the only immediate predecessor to UAP, and DRB is defined as an "*XOR-in*" task. Therefore, according to Theorem 1, Case 4, the new workflow is still well-formed and confusion-free.

So far we have demonstrated that following the cases of Theorem 1 when adding tasks to a workflow preserves its well-formedness and confusion–free properties. Again, the emphasis on this example should be on the progressive well-formedness analysis throughout the workflow construction process, instead of a reachability analysis.

Now, let us consider the dynamic modification of the workflow model. Suppose that the incident commander believes that he can also take the duty of Operation Section Chief, therefore he does not need to appoint another person for this position. Then in the workflow, tasks AOSC and RSRO are not needed. According to Theorem 3, deleting these two tasks and their related dependency arcs won't affect the well-formedness of the remaining workflow, which is partially illustrated in Fig. 7.

How about modifying the existing precedence constraints of the tasks? Suppose we want to add a new task between tasks RJAS and PIDV:

ROCH: Review organizational chart.

To do this, we first remove the dependency arc from RJAS to PIDV. According to Theorem 2, the remaining workflow is still well-formed and confusion-free. Then, we add ROCH to the workflow such that *ROCH = {RJAS} and ROCH* = {PIDV}. According to Theorem 1, the new workflow is still well-formed and confusion-free.

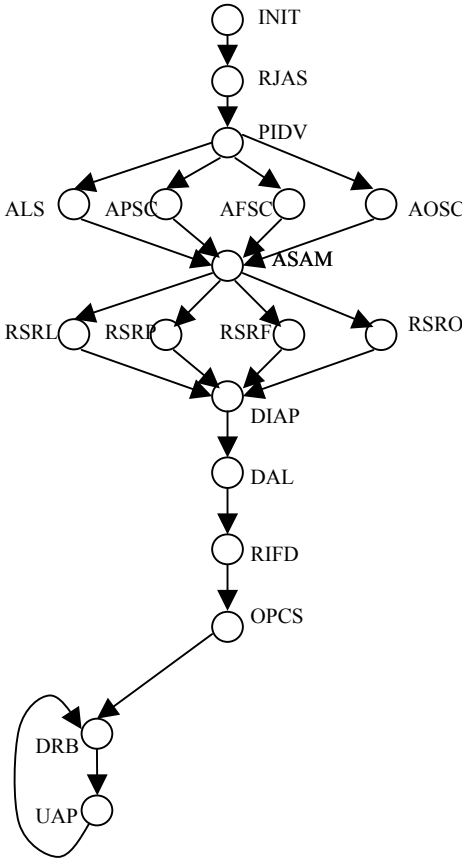

Fig. 6 Add task UAP

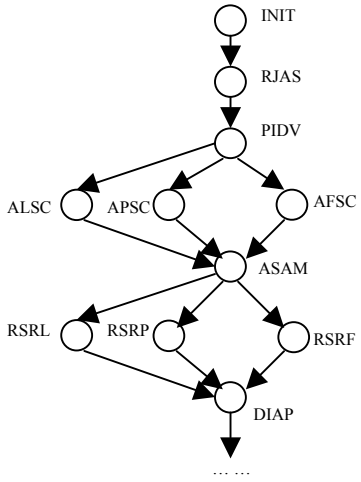

Fig. 7 Delete tasks AOSC and RSRO

We can use the same approach to model and verify workflows for other ICS components illustrated in Fig. 2. However, how to put these standalone workflow models together and how to deal with interactions between them raise compositional workflows modeling issues, which are outside the scope of this paper.

5 Concluding Remarks

In this paper we showed how to address the needs of incident command systems workflow modeling and verification using the WIFA workflow engine. We particularly addressed the dynamic factors inherent in an ICS workflow and how WIFA can be used for on-the-fly workflow modification and validation.

We have already developed a tool for WIFA and also conducted a usability assessment on the tool. The tool consists of a workflow editor, simulator and validator. The validation can be done both statically, after a workflow has been defined, or dynamically during the simulation, if a change has been introduced in the workflow. We are currently working on extending our approach to inter-organizational workflow modeling and analysis, to be able to represent the interactions between different people and organizations that need to work together for achieving different goals, particularly in dealing with an incident. This extension of the current WIFA approach will trigger the addition of data dependencies between tasks or workflows.

REFERENCES

1. van der Aalst, W.M.P. (1996) Three Good Reasons for Using a Petri Net-Based Workflow Management System, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises* (IPIC'96), pp. 179–201, Nov 1996.
2. van der Aalst, W.M.P., ter Hofstede, A.H.M., and Weske, M. (2003) Business Process Management: A Survey." *International Conference on Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1-12. Springer-Verlag, Berlin.
3. Agostini A. and DeMichelis, G. (2000) A light workflow management system using simple process models", *International Journal of Collaborative Computing* (16), pp. 335-363.
4. Attie, P. C., Singh, M. P. Sheth, A. and Rusibkiewicz, M. (1993) Specifying Interdatabase Dependencies, *Proceedings 19th International Conference on Very Large Database*, pp.134-145.
5. Casati F., Ceri S., Pernici B., Pozzi G., "Workflow evolution", *Data and Knowledge Engineering Journal*, Elsevier, vol. 24 (3), 1998, pp. 211-238.
6. Dourish, P. (2001) Process Descriptions as Organizational Accounting Devices: The Dual use of Workflow Technologies, Paper presented at GROUP'01, (ACM), Boulder, Colorado, USA
7. Ellis C. and Keddara, K. (2000) A workflow Change is a workflow", *Proceedings BPM'00*, LNCS, vol. 1806, pp. 516-534.
8. Kradolfer M., Geppert A., "Dynamic workflow schema evolution based on workflow type versioning and workflow migration", *Proceedings CoopIS'99*, Edinburgh, 1999, pp. 104-114.
9. Lawrence, P. editor, (1997) Workflow Handbook 1997, Workflow Management Coalition, John Wiley and Sons, New York.
10. Marinescu, D. C. (2002) Internet-Based Workflow Management: Towards a Semantic Web, Wiley Series on Parallel and Distributed Computing, vol. 40, Wiley-Interscience, NY.
11. Reichert M. and Dadam, P. (1998) ADEPT – supporting dynamic changes of workflows without losing control", *Journal of Intelligent Information Systems*, 10 (2), pp.93-129.
12. Rosca, D. Greenspan, S. and Wild, D. (2002) Enterprise Modeling and Decision-Support for Automating the Business Rules Lifecycle, *Automated Software Engineering Journal,* Kluwer Academic Publishers, vol.9, pp.361-404.

13. Singh, M.P., Meredith, G., Tomlinson, C., and Attie, P.C. (1995) An Event Algebra for Specifying and Scheduling Workflows, *Proceedings 4th International Conference on Database System for Advance Application*, pp. 53-60.
14. Wang, J. (1998) Timed Petri Nets: Theory and Application, Kluwer Academic Publishers.
15. Wang, J., Rosca, D., Tepfenhart, W., Milewski, A. and Stoute, M. (2005) An intuitive formal approach to dynamic workflow modeling and analysis, LNCS 3649, pp. 137-152.
16. Wang, J. and Rosca, D. (2006) Dynamic workflow modeling and analysis, the 18th International Conference on Advances Information System Engineering, Luxemborg.
17. Weske M. (2001) Formal Foundation and Conceptual Design of dynamic adaptations in a workflow management system, *Proceedings of HICSS-34*.
18. Wodtke, D. and Weikum, G. (1997) A Formal Foundation for Distributed Workflow Execution Based State Charts, *Proceedings 18th International Conference on Database Theory*.
19. Zisman, M.D. (1977) Representation, Specification and Automation of Office Procedures, *PhD thesis*, University of Pennsylvania, Warton School of Business.