Innovations Syst Softw Eng
DOI 10.1007/s11334-007-0035-y

ORIGINAL PAPER

Adding flexibility to workflows through incremental planning

Abilio Fernandes · Angelo E. M. Ciarlini ·
Antonio L. Furtado · Michael G. Hinchey ·
Marco A. Casanova · Karin K. Breitman

Received: 20 June 2007 / Accepted: 19 September 2007
© Springer-Verlag London Limited 2007

Abstract Workflow management systems usually interpret a workflow definition rigidly. However, there are real life situations where users should be allowed to deviate from the prescribed static workflow definition for various reasons, including lack of information, unavailability of the required resources and unanticipated situations. Furthermore, workflow complexity may grow exponentially if all possible combinations of anticipated scenarios must be compiled into the workflow definition. To flexibilize workflow execution and help reduce workflow complexity, this paper proposes a dual strategy that combines a library of predefined typical workflows with a planner mechanism capable of incrementally synthesizing new workflows, at execution time. This dual strategy is motivated by the difficulty of designing emergency plans, modeled as workflows, which account for real-life complex crisis or accident scenarios.

A. Fernandes · A. L. Furtado · M. A. Casanova ·
K. K. Breitman (✉)
Department of Informatics,
Pontifical Catholic University of Rio de Janeiro,
Rua Marquês de S. Vicente, 225, Rio de Janeiro,
CEP 22451-900 RJ, Brazil
e-mail: karin@inf.puc-rio.br

A. Fernandes
e-mail: abilio@inf.puc-rio.br

A. L. Furtado
e-mail: furtado@inf.puc-rio.br

M. A. Casanova
e-mail: casanova@inf.puc-rio.br

A. E. M. Ciarlini
Department of Applied Informatics,
Federal University of the State of Rio de Janeiro,
UNIRIO Av. Pasteur 458, Rio de Janeiro,
CEP 22.290-240 RJ, Brazil
e-mail: angelo.ciarlini@uniriotec.br

M. G. Hinchey
Computer Science Department, Loyola College in Maryland,
4501 N. Charles Street, Baltimore, MD 21210, USA
e-mail: mhinchey@loyola.edu

1 Introduction

Workflow management systems have been receiving considerable attention, motivated by their wide spectrum of applications. Standardization efforts are under way in the context of consortia, such as WfMC (Workflow Management Coalition), OASIS (Organization for the Advancement of Structured Information Standards), and W3C (The World Wide Web Consortium). Among other contributions, these efforts resulted in new workflow definition languages and coordination protocols. Requirements for workflow management systems comprise a long list. Among the requirements, we may highlight distributed execution, cooperation and coordination, and synchronization, which influence the way user communities work cooperatively to perform a given task.

In this paper, we introduce a notion that we define as *increasing workflow flexibility*. Briefly, workflow management systems usually interpret a workflow definition rigidly. However, there are real life situations where users should be allowed to deviate from the prescribed static workflow definition for various reasons, including lack of information and unavailability of the required resources. Furthermore, in complex domains, such as crisis response planning, designing a single workflow that accounts for all possible situations may become an overwhelming task. The final workflow may become too complex to be amenable to formal verification and may fail to cover all desired scenarios.

To achieve this increase in workflow flexibility and reduce complexity, we propose a mixed strategy. We combine a library of pre-defined workflows, of manageable complexity,

Table 1 Environmental impact of cleaning procedures for sand beaches

Cleaning procedures	Oil type				
	Type I	Type II	Type III	Type IV	Type V
ND: Natural degradation	0.00	1.00	1.00	1.00	1.00
UA: Use of absorbents	1.00	0.25	0.00	0.00	0.00
VC: Vacuum cleaning	1.00	0.00	0.00	0.00	0.00
CL: Cold, low pressure cleaning	1.00	0.00	0.00	0.25	0.25
CH: Cold, high pressure cleaning	1.00	0.25	0.25	0.25	0.25
HL: Hot, low pressure cleaning	1.00	1.00	0.50	0.50	0.50
HH: Hot, high pressure cleaning	1.00	1.00	0.50	0.50	0.50
PC: Vapor cleaning	1.00	1.00	0.75	0.75	0.75

with a planner mechanism that, based on the observed initial scenario, searches through the library, retrieves the appropriate workflows and combines them into more sophisticated units that can handle the observed scenario. We call this strategy mixed in the sense that it starts with pre-compiled plans, stored in the form of workflows, and then creates more complex plans, at run time, with the help of a planner component. This dual strategy is motivated by the difficulty of designing emergency plans, modeled as workflows, that account for real-life complex crisis or accident scenarios.

This paper is organized as follows. Section 2 expands the discussion about the motivations for our approach. Section 3 summarizes the formal framework. Section 4 exemplifies the framework. Section 5 outlines how the planning component is used. Section 6 briefly reviews work directly related to this paper. Finally, Sect. 7 contains the conclusions and describes ongoing work.

2 Motivation

An emergency plan is a collection of emergency procedures, that is, structured collections of operations that must be performed to adequately respond to specific accidental scenarios. An emergency procedure must also describe the human and material resources required, and it must include ancillary documentation, such as maps, simulation results, and lists of authorities to contact. Environmental protection agency and other government authorities indeed audit companies from time to time to verify if they have emergency plans that cover all plausible accidental scenarios, and if they have the necessary equipment or access to the required resources within a reasonable amount of time. Similarly, plans exist for recovery of NASA spacecraft following incidents, as do plans for rescuing astronauts during spacewalks or from the International Space Station (ISS).

For example, consider an emergency plan for an oil terminal located in a harbor facility. The following are possible accident scenarios: vessels docked at the harbor may catch fire, causing explosions followed by oil spills; vessels may hit underwater oil pipelines causing oil spills of a different nature; in-land oil tanks may catch fire and produce harmful pollution reaching nearby communities.

The plan must include, for example, specific procedures for cleaning coastal areas affected by an oil spill. The procedures typically take into account the oil type and the characteristics of the coastal area. Table 1 provides a schematic example of cleaning procedures, where the type of coastal area is a Sand Beach and the oil types are named Type I–Type V. The weights in table cells indicate the environmental impact of each procedure: 0.00 indicates the smallest environmental impact, 0.25 some impact, 0.50 a significant impact, 0.75 the greatest impact, and 1.00 inapplicable.

Now, suppose that an emergency team is assigned to an accident. The team will be referred to as the *user* of the emergency plan in what follows. Suppose that the user comes to a point in the overall emergency plan execution where he needs to select a cleaning procedure for a sand beach affected by an oil spill. The user can then look up in Table 1 to select the best procedure to clean the beach. For example, if the oil is of Type II, the best procedures are “VC: Vacuum cleaning” and “CL: Cold, low pressure cleaning”. Based on the equipment available, he may then select a feasible procedure and proceed to execute it. This sequence of steps has to be repeated for each coastal segment that the oil spill affected. Note that the cleaning procedures selected for the various coastal segments may interfere with each other, if they use the same equipment or if the personnel available are limited. The user would then have to schedule the cleaning procedures within the bounds of the available resources.

Emergency response information systems [1–3] are information systems designed to help automate emergency plans and expedite access to the required data during accidents. Most of the work in this paper is motivated by the development of InfoPAE [4,5], an emergency response information system that is currently operational in hundreds of oil facilities. InfoPAE combines a workflow management system and a geographic database, and is based on the translation

of very large (paper) emergency plans into equally complex workflows. However, an analysis of the complexity of the emergency plans suggested that the workflows should be synthesized on demand, based on the current accidental scenarios, using a library of reasonably small workflows that address simple scenarios. Early experiments confirmed that this is indeed a feasible strategy.

The above example covers just a simple scenario—an oil spill. The problem lies in that an accident may trigger a sequence of other events, such as an explosion, followed by fire and by an oil spill. Therefore, emergency plans must not be too rigid or else they become too complex to accommodate all possible scenarios, generated by multiple events. Furthermore, as already pointed out, the amount of resources available—equipment, personnel, etc.—constrains the number of procedures that can be executed in parallel.

With this background motivation, we propose a strategy to help reduce workflow complexity and flexibilize workflow execution. The strategy combines a library of predefined typical workflows, of manageable complexity, with a planner capable of synthesizing more complex workflows, at execution time, tuned to the scenario in question. We call this strategy *incremental* or *online planning*. Note that it potentially reduces workflow complexity, because it replaces the need to anticipate all accidental scenarios into a static workflow by on-the-fly plan generation.

A key element of the strategy is the notion of a *condition goal inference rule* (CGIR) that provides the planner with enough information to incrementally synthesize partial plans. Indeed, planners are typically given initial and final conditions, but little or no information that helps guide the planning process in between.

Anticipating the discussion in Sect. 4, we illustrate the use of CGIRs in the context of our running example. We consider a simplified model with just two types of incidents: (1) fire on facilities, such as oil pipelines and oil tanks; (2) oil spills that pollute shoreline segments, such as sand beaches and reefs. The emergency procedures we adopt as examples are highly stylized and follow simple steps, summarized in Figs. 1 and 2. The Clean step of Fig. 2 is refined according to Table 1.

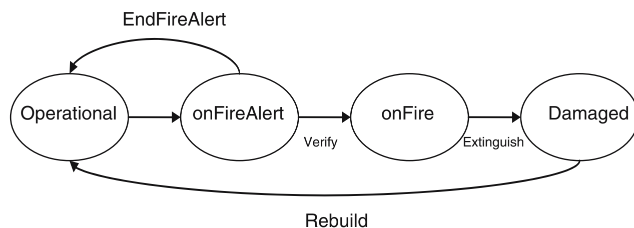

Fig. 1 States of a facility

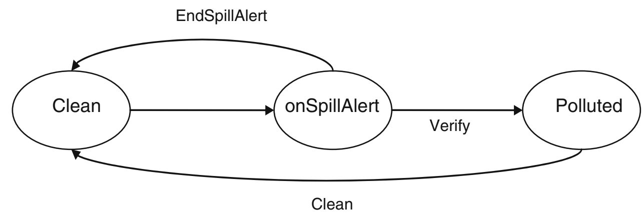

Fig. 2 States of a shoreline segment

Consider the emergency procedure, depicted in Fig. 1, for any facility F that may catch fire. If the fire alarm sounds for F, the status of F should be changed from *Operational* to *OnFireAlert* and should remain so until confirmation. From the *OnFireAlert* state, either the fire is confirmed and procedures to extinguish it must be started, or F goes back into operation.

If facility F reaches the *OnFireAlert* state, the following condition goal inference rules force the planner to bring F back to the *Operational* state (see Sect. 4 for a formal statement of these examples):

CGIR1. (if F is *OnFireAlert* then eventually (F is *OnFire* or F is *Operational*)) always holds

CGIR2. (if F is *OnFire* then eventually (F is *Damaged*)) always holds

CGIR3. (if F is *Damaged* then eventually (F is *Operational*)) always holds

In real-life situations, the emergency plan should also include operations to isolate each facility G in the vicinity of F, when F becomes on fire alert. The facility G should be forced into the *OnFireAlert* state and remain so (or actually be *OnFire* or *damaged,* that is, not in the *operational* state) until F goes back to the *operational* state. Figure 3 illustrates this situation.

We note that, differently from F, which reached the *OnFireAlert* state as a consequence of a fortuitous event, G is forced into this state by the planner. This transition is guaranteed by the following condition goal inference rule:

CGIR4. (if F is *OnFireAlert* and G is a facility in the vicinity of F and G is *Operational* then eventually (G is *OnFireAlert*)) always holds

Furthermore, when F is located near the shore, the planner also has to take into consideration possible oils spills and put any nearby shoreline segment S on spill alert. Figure 4 illustrates this situation.

Again, as a response to a fortuitous event that affected facility F, the planner has to force a state transition of any nearby shoreline segment S. This transition is guaranteed by the following condition goal inference rule:

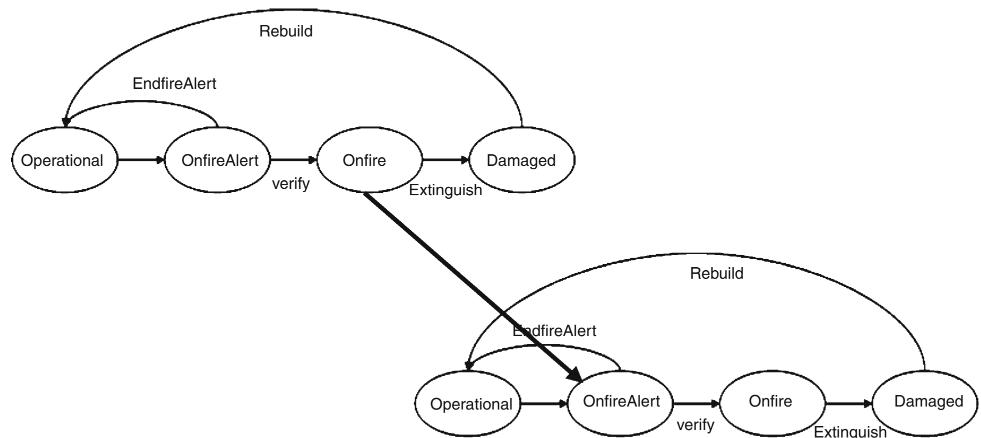

Fig. 3 Interaction between facility f1 and a nearby facility f2

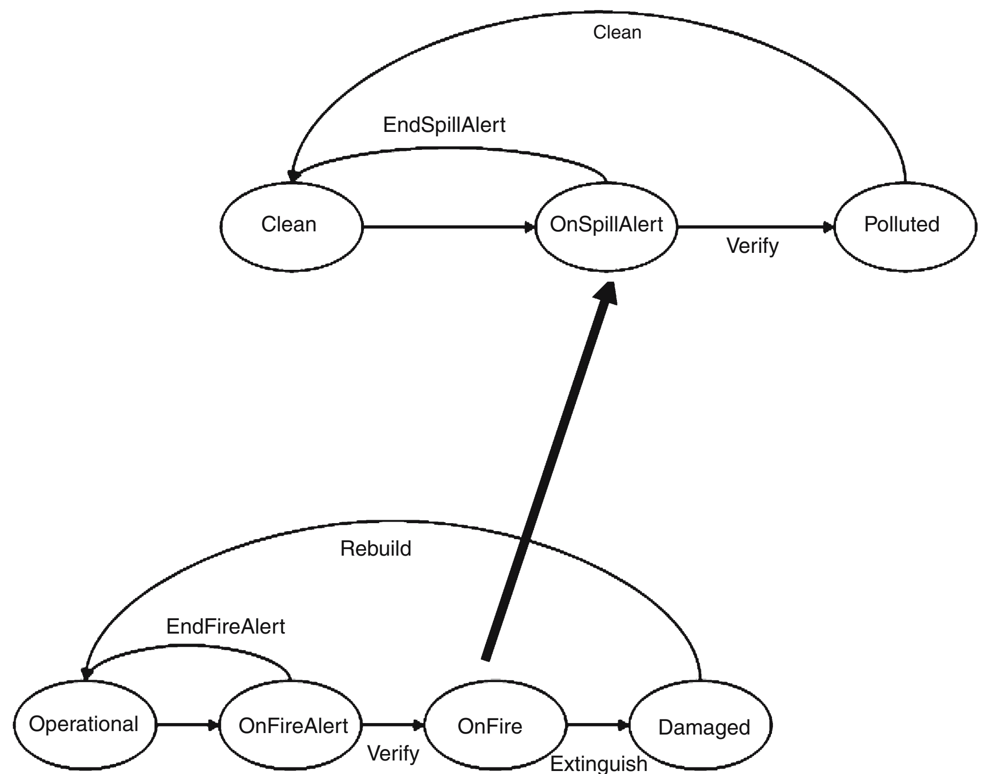

Fig. 4 Interaction between facility f1 and a nearby shoreline segment sl1

CGIR 5. (if F is *OnFireAlert* and S is a shoreline segment in the vicinity of F and S is *clean* then eventually (S is *OnSpillAlert*)) always holds

Therefore, as a consequence of a fire alert in F, the planner must respond with operations that:

1. Extinguish the fire on facility F (part of the regular workflow for F).
2. Force a change of status of any facility G within the facility danger perimeter of F.
3. Force a change of status of any shoreline segment S within the shoreline danger perimeter of F.

The above discussion ultimately means that facilities should go back to the *operational* state and shoreline segments to the *clean* state. In a strict sense, the workflow for either a facility or a shoreline segment could be reduced to a single goal—return to the *operational* or the *clean* state. Of course, it does not provide much information to the planner on how to do so. That is, the role played by the condition goal inference rules is to generate intermediate goals for the planner. During plan processing, undesirable states may be reached, thus forcing the planner to (re)apply additional CGIRs until the *operational* (or *clean*) state is reached.

Indeed, a conventional planner would consider every transition from the initial to the final state. What we are

proposing is slightly different, as we expect the planner to start up from any (undesirable) state and move forwards, prompted by condition goal inference rules.

Thus, our immediate problem is reduced to verifying that we have defined enough condition goal inference rules that guarantee that, eventually, each facility is brought back to the *Operational* sate, and that each shoreline segment reaches the *Clean* state.

3 Formal framework

The formal framework follows Ciarlini et al. [6] and specializes familiar concepts to meet the restrictions of the planner component described in Sect. 5.

We define a *planning language* as a many-sorted first-order language, with equality, with a set of special function symbols, the *operation names,* and a set of predicate symbols, the *database predicate symbols*. In what follows, let $\mathcal{L}$ be a planning language, $\mathcal{O}$ be the set of operation names of $\mathcal{L}$, and $\mathcal{D}$ be the set of database predicate symbols of $\mathcal{L}$. The syntax and semantics of $\mathcal{L}$ follow as usual, so that we will attain just to the concepts that are motivated by the planner. A (constant) *substitution* is a function i mapping each variable of $\mathcal{L}$ to a constant (of the same sort). An *expression* is a formula (of various forms, defined in what follows) or a term. Given an expression e, we use $e[i]$ for the *substituted version* of e.

A *structure* M for $\mathcal{L}$ assigns a meaning to each symbol of $\mathcal{L}$. Given a formula σ and a set of formulae Σ of $\mathcal{L}$, we use $\models_M \sigma$ to denote that M *satisfies* σ, or that σ *holds* in M, and $\Sigma \models \sigma$ to denote that σ is a *logical consequence* of Σ.

We remark that M should assign the standard meaning to the familiar function and predicate symbols, such as "+" and "<", which should not be included in the set of operation names and database predicate symbols. This remark becomes important in the context of the planner, described in Sect. 5, which is implemented based on this assumption.

A *database literal* is a literal whose predicate symbol is in $\mathcal{D}$, and a *database fact* is a positive ground database literal. A *possible fact* of M is a database fact of $\mathcal{L}$ that holds in M. A *possible database state* of M is a set of possible facts of M. We define the notion of *holding* at a possible database state s as follows:

- For a database fact F, we define that F *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \models_M F$, iff $F \in \boldsymbol{s}$.
- For a database literal L, we define that L *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \models_M L$, iff $\boldsymbol{s} \models_M G$ for every ground instance G of L.
- For a conjunction K of database literals, we define that K *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \models_M K$, iff $\boldsymbol{s} \models_M L$ for every L that occurs in K.
- For a set $\boldsymbol{L}$ of database literals, we define that $\boldsymbol{L}$ *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \models_M \boldsymbol{L}$, iff $\boldsymbol{s} \models_M L$ for every $L \in \boldsymbol{L}$.

We model operations in terms of their pre- and post-conditions [6]. Pre-conditions are conjunctions of (positive or negative) literals that must hold at the state in which the operation is to be executed. Post-conditions, or effects, consist of two sets of literals: those to be asserted and those to be retracted as a consequence of executing the operation. More precisely, an *operation* is a tuple $e = (o, d, P, Q)$ where

- o is an operation name
- d is a term of the form $o(t_{1}, \ldots, t_n)$, called the *input declaration* of o.
- P is a set of database literals, specifying the *pre-conditions* of e.
- Q is a set of database literals, specifying the *post-conditions* of e.

The operation $e = (o, d, P, Q)$ is *terminal* iff

- The declaration d is a variable-free term.
- The post-conditions Q are ground.
- The positive pre-conditions are ground

Negative pre-conditions in terminal operations are not restricted to be ground. Variables might occur, such as in $\neg$ p(x), to express that $\forall x \neg$ p(x).

The structure M assigns a relation o^M (respecting sorts) to each operation symbol o. A *possible operation call* in M is a term of the form $o(c_{1}, \ldots, c_n)$, where $c_1, \ldots, c_n$ are constants, such that
$< c_1^M, \ldots, c_m^M > \in o^M$.

The *meaning* of a terminal operation $e = (o, d, P, Q)$ in M is the set e^M consisting of all pairs $(\boldsymbol{s}, \boldsymbol{t})$ of possible database states such that the following holds true:

- $\boldsymbol{s} \models_M P$ (pre-conditions are satisfied).
- $\boldsymbol{t} \models_M Q$ (post-conditions are satisfied).
- for every possible database fact f of M, if $f \notin Q$ and $\neg f \notin Q$, then $\boldsymbol{s} \models f$ iff $\boldsymbol{t} \models f$ (which is the frame requirement: preservation of satisfaction from $\boldsymbol{s}$ to $\boldsymbol{t}$ for ground database literals that are neither established nor negated by the post-conditions in Q).

Therefore, the domain of e^M consists of the states satisfying the pre-conditions of e, and e^M transforms each state $\boldsymbol{s}$ in its domain to a state t satisfying the post-conditions of e. Given an operation e, we use $e[i]$ for the *substituted version* of e, obtained by applying i to the declaration, the pre- and the post-conditions of e. The *meaning* of a non-terminal operation e in M is the set e^M consisting of all pairs $(\boldsymbol{s}, \boldsymbol{t})$

of possible database states such that $(\boldsymbol{s}, \boldsymbol{t}) \in e[i]^M$, for some substitution i.

A *temporal database* of M is a pair $T = (\boldsymbol{S}, \boldsymbol{O})$ consisting of a sequence $\boldsymbol{S}$ of possible database states of M and a, possibly empty, sequence $\boldsymbol{O}$ of terminal operations such that

- $|\boldsymbol{S}| = |\boldsymbol{O}| + 1$ (the length of $\boldsymbol{S}$ is the length of $\boldsymbol{O}$ plus one)
- $< \boldsymbol{S}_i, \boldsymbol{S}_{i+1} > \in \boldsymbol{O}_i^M$, for each $\imath \in [1, |\boldsymbol{O}|]$ (non-initial states are caused by terminal operations)

Note that, if $\boldsymbol{O}$ is empty, then T reduces to just one database state. Given a temporal database $T = (\boldsymbol{S}, \boldsymbol{O})$ of M, and $k \in [1, |\boldsymbol{S}|]$, the kth *continuation* of T is the temporal database $T^k = (\boldsymbol{S^k}, \boldsymbol{O^k})$ such that $\boldsymbol{S^k}$ and $\boldsymbol{O^k}$ are the suffixes of $\boldsymbol{S}$ and $\boldsymbol{O}$ starting on the kth element of $\boldsymbol{S}$ and $\boldsymbol{O}$, respectively. Note that a continuation $T^{\boldsymbol{k}} = (\boldsymbol{S^k}, \boldsymbol{O^k})$ of T is indeed a temporal database, and that the temporal database consisting of just the last state of T and the empty sequence of operations is the nth continuation of T, where $n = |\boldsymbol{S}|$.

An *atomic temporal formula* is an expression of one of the following (restricted) forms, where $\boldsymbol{L}$ and $\boldsymbol{K}$ are conjunctions of database literals, or a single database literal:

$\Diamond \boldsymbol{L}$	$\boldsymbol{L}$ eventually holds
$\Box\mathbf{L}$	$\boldsymbol{L}$ always holds
$\bigcirc \boldsymbol{L}$	$\boldsymbol{L}$ holds next
$\boldsymbol{L}\mathrm{U}\boldsymbol{K}$	$\boldsymbol{L}$ holds until $\boldsymbol{K}$

More complex temporal formulae are introduced as usual. We define the notion of *holding* at a temporal database $T = (\boldsymbol{S}, \boldsymbol{O})$ of M, as follows [6]:

- For a conjunction $\boldsymbol{L}$ of database literals, we define that $\boldsymbol{L}$ *holds* in T for M, denoted $T \models_M \boldsymbol{L}$, iff $\boldsymbol{S}_1 \models_M \boldsymbol{L}$ ($\boldsymbol{L}$ holds in the first state $\boldsymbol{S}_1$ of T).
- For a formula of the form $\Diamond \boldsymbol{L}$, we define that $\Diamond \boldsymbol{L}$ *holds* in T for M, denoted $T \models_M \Diamond \boldsymbol{L}$, iff for some $\mathrm{k} \in [1, |\boldsymbol{S}|]$, $T^k \models_M \boldsymbol{L}$ ($\boldsymbol{L}$ holds in some continuation T^k of T).
- For a formula of the form $\Box \boldsymbol{L}$, we define that $\Box \boldsymbol{L}$ *holds* in T for M, denoted $T \models_M \Box \boldsymbol{L}$, iff for all $\mathrm{k} \in [1, |\boldsymbol{S}|]$, $T^k \models_M \boldsymbol{L}$ ($\boldsymbol{L}$ holds in all continuations T^k of T).
- For a formula of the form $\bigcirc \boldsymbol{L}$, we define that $\bigcirc \boldsymbol{L}$ *holds* in T for M, denoted $T \models_M \bigcirc \boldsymbol{L}$, iff $T^2 \models_M \boldsymbol{L}$ ($\boldsymbol{L}$ holds in T^2, the continuation starting on the second state of T).
- For a formula of the form $\boldsymbol{L}\mathrm{U}\boldsymbol{K}$, we define that $T \models_M (\boldsymbol{L}\mathrm{U}\boldsymbol{K})$ *holds* in T for M, denoted $T \models_M (\boldsymbol{L}\mathrm{U}\boldsymbol{K})$, iff there is $\mathrm{p} \in [1, |\boldsymbol{S}|]$ such that $T^p \models_M \boldsymbol{K}$ and, for all $\mathrm{q} \in [1, \mathrm{p})$, $T^q \models_M \boldsymbol{L}$ ($\boldsymbol{L}$ holds in all continuations until reaching, but excluding, a continuation where $\boldsymbol{K}$ holds).

We introduce more complex temporal formulae and accordingly extend the notion of holding at a temporal database as usual.

A *condition goal inference rule* is recursively defined as follows:

- A temporal formula of the form $\Diamond \boldsymbol{L}$ is a condition goal inference rule, where $\boldsymbol{L}$ is a conjunction of database literals, or a single database literal.
- A temporal formula of the form $(\boldsymbol{L} \Rightarrow \Gamma)$ or of the form $\Box(\boldsymbol{L} \Rightarrow \Gamma)$ is a condition goal inference rule, where $\boldsymbol{L}$ is a conjunction of database literals, or a single database literal, and Γ is a condition goal inference rule.

For example, the formula

$$\Box(C_1 \Rightarrow \Box(C_2 \Rightarrow \Box(C_3 \Rightarrow \Diamond G)))$$

is a condition goal inference rule. It holds at a temporal database $T = (\boldsymbol{S}, \boldsymbol{O})$ of M iff, whenever there are three states $\boldsymbol{S}_{c1}, \boldsymbol{S}_{c2}, \boldsymbol{S}_{c3}$, with $c_1 \leq c_2 \leq c_3$, such that $\boldsymbol{C_i}$ holds in $\boldsymbol{S}_{ci}$, for $i = 1, 2, 3$, then there must be a state $\boldsymbol{S}_g$ with $c_3 \leq g$, such that $\boldsymbol{G}$ holds in $\boldsymbol{S}_g$. Intuitively, $\boldsymbol{C_1}, \boldsymbol{C_2}, \boldsymbol{C}_3$ define a sequence of conditions that, if satisfied, implies that the goal $\boldsymbol{G}$ must be satisfied in a future state (with respect to the state where $\boldsymbol{C_3}$ is satisfied).

A *plan* is a triple $P = (\boldsymbol{L}, O, p)$ where

- $\boldsymbol{L}$ is a set of literals, capturing the *constraints* and *the initial database state* of P
- O is a set of operations
- (O, p) is a directed acyclic graph, that is, p is a partial order on O

A plan P is *terminal* iff all operations of P are terminal. The *meaning* of a terminal plan $P = (\boldsymbol{L}, O, p)$ in M is the set P^M consisting of all temporal databases $T = (\boldsymbol{S}, \boldsymbol{O})$ of M, such that

- $\boldsymbol{S}_1 \models_M \boldsymbol{L}$
- $\boldsymbol{O}$ is a sequence consisting exactly of the operations in O in an order consistent with the partial order p

The *meaning* of a non-terminal plan $P = (\boldsymbol{L}, O, p)$ for M is the set P^M consisting of all temporal databases $T \in P[i]^M$, for all substitutions i of all variables occurring in operations in O. A temporal formula Γ *holds* in a plan $P = (\boldsymbol{L}, O, p)$ for M, denoted $P \models_M \Gamma$, iff $T \models_M \Gamma$, for all $T \in P[i]^M$, for all substitutions i of all variables occurring in operations in O.

A *workflow* is defined very similarly to a plan, except that it has special operations that model a minimal set of model workflow control constructs [7]. A *workflow* is a triple $W = (\boldsymbol{K}, Q, r)$ where

- $\boldsymbol{K}$ is a set of literals, the *constraints* of W

- Q is a set of operations, including the *split,* the *join* and the *choice* operations
- (Q, r) is a directed acyclic graph with a source node, called the *start* node (or *start* operation)

A workflow defines a set of embedded plans in the following sense. A plan $P = (\boldsymbol{L}, O, p)$ is *embedded* in a workflow $W = (\boldsymbol{K}, Q, r)$ iff

- $\boldsymbol{K} \models_M \boldsymbol{L}$
- (O, p) is a sub-graph of (Q, r) such that
  - O contains the start node of W
  - Each choice operation in O contains only one successor
  - For each arc (n, m) in r such that n is a non-choice operation, if n in O, then (n, m) is also in p

The semantics of split and join operations are already reflected in the definition of plan, when we allowed parallel operations. The semantics of the choice operation should intuitively be that the pre-conditions of the successor of a choice operator must be valid. It is captured indirectly by defining that a plan embedded in a workflow is a *valid execution* in M for W or, simply, is *valid* in M for W iff the set P^M is not empty.

Finally, a plan P is *consistent* with a set Σ of temporal formulae with respect to a structure M iff, for every $\sigma \in \Sigma$ for every $T \in P^M$, we have $T \models_M \sigma$. A workflow W is *consistent* with a set Σ of temporal formulae with respect to a structure M iff, for every $\sigma \in \Sigma$, for every plan P embedded in W that is valid in M for W, for every $T \in P^M$, we have $T \models_M \sigma$.

4 Formalization of the example

To formalize the example of Sect. 2, we first introduce the following language:

$f_1, f_2, \ldots$	Constants denoting facilities
$a_1, a_2, \ldots$	Constants denoting shoreline segments
$OilPipeline, oilTank, \ldots$	Constants denoting facility types
$SandBeach, reefs, \ldots$	Constants denoting shoreline segment types
Type I, type II, type III, type IV, type V, undefined	Constants denoting oil types
$s(X) = Y$	Y is the state of X (see Figs. 1, 2)
$next(X, Y)$	Y is a successor state of X (see Fig. 1 and Fig. 2)
$t(L) = T$	Location L is of type T
$p(L) = T$	Location L is polluted with oil type T
$location(F) = L$	L is the location of facility F
$shutdown(F)$	Facility F is shut down
$isolated(L)$	Location L is isolated
$facilityWithin\text{-}DangerPerimeter(F, G)$	Facility G is within the danger perimeter of facility F (with respect to a fire on F)
$shorelineWithin\text{-}DangerPerimeter(F, S)$	Shoreline segment S is within the danger perimeter of facility F (with respect to a fire on F)

The following are the static constraints:

FC1. $\Box(s(F) = Operational \Rightarrow (\neg shutdown(F) \wedge \neg isolated(F)))$
(If a facility F is operational, then it is neither shutdown nor isolated.)

FC2. $\Box(p(L) = T \Rightarrow (T = typeI \vee T = typeII \vee T = typeIII \vee T = typeIV \vee T = typeV \vee T = undefined))$
(The valid oil types are $typeI$ through $typeV$)

SC1. $\Box(s(L) = Clean \Rightarrow \neg isolated(L))$
(If a shoreline segment L is clean, then it is not isolated.)

The *condition goal inference rules* force the planner to select operations that will provoke state transitions. The *facility condition goal inference* rules guarantee that transitions from states *OnFireAlert, OnFire* and *Damaged* will indeed occur and will eventually bring a facility back to the *operational* state, but they need not guarantee that the facility will eventually move out of the *Operational* state:

GF1. $\Box(s(F) = OnFireAlert \Rightarrow \Diamond(s(F) = X \wedge next(OnFireAlert, X)))$
GF2. $\Box(s(F) = OnFire \Rightarrow \Diamond(s(F) = X \wedge next(OnFire, X)))$
GF3. $\Box(s(F) = Damaged \Rightarrow \Diamond(s(F) = X \wedge next(Damaged, X)))$

(Note that GF2 and GF3 could be simplified since *OnFire* and *Damaged* both have just one successor state)

The interaction between a facility and those on its danger perimeter is formulated as:

GF4. $\Box((s(F) = OnFireAlert \wedge facilityWithin\text{-}DangerPerimeter(F, G) \wedge s(G) = Operational) \Rightarrow \Diamond(s(G) = OnFireAlert))$

The interaction between a facility and shore line segments on its danger perimeter is formulated as:

GF5. $\square((s(F) = OnFireAlert \wedge shorelineWithinDangerPerimeter(F, S) \wedge s(S) = Clean) \Rightarrow \lozenge(s(S) = OnSpillAlert))$

Additional *facility CGIRs* work as temporal restrictions that force the planner to schedule further operations:

GF6. $\square(s(F) = OnFire \Rightarrow \lozenge(shutdown(F) \wedge isolated(location(F))))$

(If a facility F becomes on fire, then F must eventually be shut down and the area isolated).

The *shoreline segment condition goal inference* rules guarantee that transitions from states OnSpillAlert and polluted will indeed occur and will eventually bring a shoreline segment back to the clean state, but they need not guarantee that the shoreline segment will eventually move out of the clean state:

GS1. $\square(s(L) = OnSpillAlert \Rightarrow \lozenge(s(F) = X \wedge next(OnSpillAlert, X)))$.

GS2. $\square(s(L) = Polluted \Rightarrow \lozenge(s(F) = X \wedge next(Polluted, X)))$.

(Again note that GS2 could be simplified since *polluted* has just one successor state).

Additional *shoreline CGIRs* work as temporal restrictions that force the planner to schedule further operations:

GS3. $\square(s(L) = Polluted \Rightarrow \lozenge(isolated(L)))$.

(If a shoreline segment L becomes polluted, then L must eventually be isolated).

Operations are specified as follows, where $\{P\}o\{Q\}$ indicates that P are the pre-conditions and Q are the post-conditions of operation o.

The following are operations on facilities:

FO1. $\{s(F) = OnFireAlert\}EndFireAlert(F)\{s(F) = Operational\}$

FO2. $\{s(F) = OnFireAlert\}Verify(F)\{s(F) = OnFire\}$

FO3. $\{s(F) = OnFire\}Extinguish(F)\{s(F) = Damaged\}$

(Operations required to enforce GF1 to GF2, that is, the state transitions as per Fig. 1).

FO4. $\{s(F) = Damaged\}\ Rebuild(F)\{s(F) = Operational \wedge \neg shutdown(F) \wedge \neg isolated(location(F))\}$

FO5. $\{s(F) = OnFire\}Shutdown(F)\{shutdown(F) \wedge isolated(location(F))\}$

(Operation required to enforce GF6)

FO6. $\{s(F) = Operational\}SetOnFireAlert(F)\{s(F) = OnFireAlert\}$

(Operation required to document fortuitous fire incident).

The following are the operations on shoreline segments:

SO1. $\{s(L) = OnSpillAlert\}EndSpillAlert(L)\{s(L) = Clean\}$

SO2. $\{s(L) = OnSpillAlert\}Verify(L)\{s(L) = Polluted\}$

(Operations required to enforce GS1, that is, some of the state transitions as per Fig. 1).

SO3. $\{s(L) = Polluted\}Analyze(L)\{\neg p(L) = undefined\}$.

(Operation required to set a precondition for the cleaning operations).

SO4. $\{s(L) = Clean\}SetOnSpillAlert(L)\{s(L) = OnSpillAlert\}$.

(Operation required to document fortuitous oil spill incident).

The clean operation may be refined as per Table 1 (without considering weights for the time being)

ND1. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeII)\}NaturalDegradation(L)\{s(L) = Clean\}$.

ND2. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeIII)\}NaturalDegradation(L)\{s(L) = Clean\}$.

ND3. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeIV)\}NaturalDegradation(L)\{s(L) = Clean\}$.

ND4. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeV)\}NaturalDegradation(L)\{s(L) = Clean\}$.

UA1. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeI\}UseOfAbsorbents(L)\{s(L) = Clean\}$.

UA2. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeII\}UseOfAbsorbents(L)\{s(L) = Clean\}$.

VC. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeI\}VacuumCleaning(L)\{s(L) = Clean\}$.

CL1. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeI\}$. $ColdLowPressureCleaning(L)\{s(L) = Clean\}$

CL2. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeIV\}.$
$ColdLowPressureCleaning(L)\{s(L) = Clean\}$

CL3. $\{s(L) = Polluted \wedge t(L) = sandbeach \wedge p(L) = typeV\}.$
$ColdLowPressureCleaning(L)\{s(L) = Clean\}$

CH. $\{s(L) = Polluted \wedge t(L) = sandbeach\}ColdHighPressureCleaning(L)\{s(L) = Clean\}.$

HL. $\{s(L) = Polluted \wedge t(L) = sandbeach\}HotLowPressureCleaning(L)\{s(L) = Clean\}.$

HH. $\{s(L) = Polluted \wedge t(L) = sandbeach\}HotHighPressureCleaning(L)\{s(L) = Clean\}.$

PC. $\{s(L) = Polluted \wedge t(L) = sandbeach\}VaporCleaning(L)\{s(L) = Clean\}.$

(Since the preconditions of an operation have to be a conjunction of literals, ND1 to ND4 cannot be combined into a single definition; the same observation applies to UA1 and UA2, and to CL1, CL2 and CL3).

5 The planner module

We are experimenting with a planner component that has been successfully used in the interactive plot generator (IPG) system [8]. IPG is an implementation of a logical framework for the simulation of the interactions of agents [9]. It has been applied both for decision support and for generating stories as part of an interactive storytelling tool [10]. In this section, we briefly describe IPG and show how it can be used to compose workflows.

5.1 An overview of IPG

The IPG system consists of a planner integrated with a plan-recognition mechanism and goal-inference methods. IPG has been developed in Prolog, using constraint logic programming features. The planner is hierarchical and non-linear (i.e., it generates partially ordered plans) and was adapted from Abtweak [11]. The IPG planner logic uses a syntactic structure for specifying and reasoning about the possible temporal databases that can be created from an initial database state by the execution of a (partially ordered) set of operations. The temporal reasoning adapts the modal truth criterion (MTC) [12] to a database context conforming to the closed world assumption (CWA) [13].

IPG agents' behavior is modeled by means of a knowledge base composed of an initial database, condition goal inference rules (CGIRs), operations and integrity constraints. At the behavioral level, condition goal inference rules are specified to describe the situations that generate new goals the agents should pursue. At the functional level, operations describe the possible actions the agents can perform to achieve their goals. Since operations and goal inference rules are defined for the various classes of agents, IPG simulates their interaction and the planner generates the possible plans of narratives that may occur.

The simulation process occurs in multiple stages of user intervention, goal-inference and planning. The user intervenes in the process by adding observations. Observations are used to specify operations to be artificially inserted, artificial goals, describing situations that should be forced, and additional constraints on the execution order of the operations.

At the beginning of a simulation, the user may optionally specify initial observations. Figure 5 illustrates the initial state of the IPG planner with an implementation for a fire alert situation, as described in Sect. 2 of this paper.

Based on the initial database, CGIRs are used to infer goals. To create a combination of operations, conforming to the observations and able to achieve the goals, the planner is applied: operations are inserted in the plan and constraints on their order and on the value of variables are imposed. When a coherent plan (i.e., a plan in which all pre-conditions necessarily hold at their execution time) is obtained, the current planning phase is finished and another user intervention occurs: the current plan is presented to the user, who can reject it and ask for another solution, accept it as it is or modify it, by adding new observations. To illustrate the planner in action, we apply CGIR5 (Sect. 2) to the initial situation depicted in Fig. 5. As a consequence to the fire alert situation, the planner will extinguish the fire on facility FC01 and force the change of status on a facility FC02, located within the perimeter of FC01. Figure 6 illustrates the sequence of operations devised by IPG.

After user intervention, new goal-inference and planning phases occur. The new operations inserted during the last planning phase may lead to the inference of new goals to be fulfilled at the planning phase. At the planning phase,

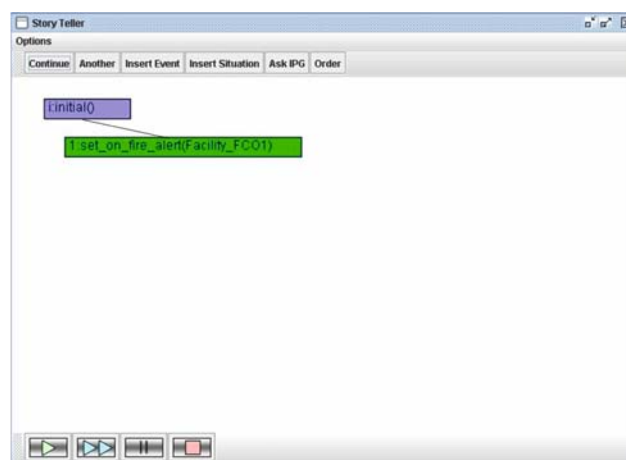

Fig. 5 Initial state of the IPG planner

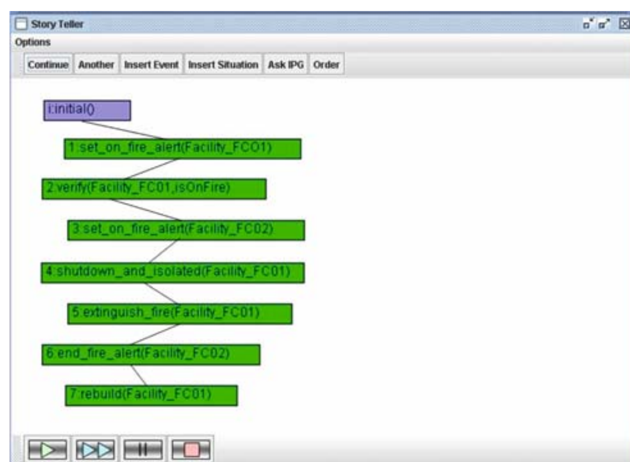

Fig. 6 IPG in action: response plan to fire alert situation

the planner copes then with both new inferred goals and the observations added by the user in the last intervention. When a partial coherent plan is generated, the simulation continues with new user intervention and the inference of new goals. Thus, we have a progressive process, in which a plan is constructed on the fly. The simulation is finished when no new goal is inferred during a goal-inference phase and the user does not add any observation. During planning phases, integrity constraints are checked to discard alternatives that violate them.

CGIRs try to capture the problems and opportunities emerging at possible states of a partial plan. They are conditionals which are exhaustively evaluated: whenever the antecedent holds but not the consequent, the consequent is taken as an observation to be added to the plan before the next planning stage. Notice that, at the start of the simulation, both the observations informed by the user and those that can be inferred from an initial plan, containing only the initial database state, are added.

IPG is fully compatible with the formal framework described in Sect. 3. CGIRs, operations and integrity constraints of the example described in Sect. 4, for instance, can be directly translated to IPG's formalism. If we have only atomic operations, the generation of emergency plans by means of IPG is straightforward. If the number of operations is large, however, the process might be time consuming.

An alternative way to obtain plans for achieving the goals of an agent is to take, from an adequately structured library, a pre-existing *typical workflow* or *plan,* adapting it if necessary to specific circumstances. Taking typical plans as building blocks corresponds, in artificial intelligence terminology, to start from commonly used *scripts,* rather than constructing new plans from primitive operations [14,15]. The composition of workflows for the generation of emergency plans with IPG is explained in the sequel.

5.2 Workflow composition

Workflows basically avoid the construction of new plans always from primitive operations. Intuitively, a workflow describes a set of plans that are frequently used for a given set of scenarios. The operations in the example of Sect. 4 were introduced as atomic operations to simplify the explanation about the possible use of CGIRs combined with a planning algorithm. In fact, most of them are workflows with various atomic operations, which have to be combined. An example is the operation "shutdown and isolate facility", illustrated in Fig. 6, that can be understood as a separate workflow. Shutting down and isolating a facility comprise a series of moderately complex operations that include shutting power, water and gas circuitry, closing valves, putting in place evacuation plans and securing the property.

In order to combine small workflows, we encapsulate them as operations with basic pre-conditions and post-conditions. The basic pre-conditions describe, for each workflow, the most important necessary conditions for its execution. The basic post-conditions describe the main effects of executing the workflow. Based on such descriptions, IPG can insert workflows, during the planning phase, as if they were atomic operations. In this way, the treatment of emergencies in complex systems would not demand the previous creation of huge workflows, replicating various parts of emergency procedures. CGIRs can guide the process, composing workflows in a compatible way with the initial situation described in the initial database state. If a usual emergency situation occurs, the planner tends to select predefined workflows to reach the goals inferred by CGIRs, because this option is computationally cheaper than composing a new plan from scratch. When there is no workflow ready to reach a goal, the planner can still try to combine workflows and/or atomic operations.

The problem with this approach is that pre-conditions and post-conditions of a workflow are not deterministic, because the workflow might have operations of type choice and the order of its basic operations may vary. As a consequence, we cannot guarantee that the global plan generated so far is executable. To validate the plan, we then extract the plans embedded within each workflow. Such plans replace the corresponding workflow in the global plan, so that a global flat plan, containing only atomic operations, is obtained. Notice that the flattening process is also necessary for the inference of new goals at the next simulation stage.

To obtain flat plans, there is a refinement planning phase. This phase initially simulates the possible executions of operations of type choice, within each workflow, and generates all possible combinations of options. Each combination corresponds to an alternative flat plan, competing to be the solution. For each alternative, the planning algorithm checks whether all pre-conditions of all operations are already

fulfilled. If it is not the case, the planner can still adapt the alternative, adding new atomic operations and constraints on the execution order of the operations and on the value of the variables.

For the time being, our workflows are only acyclic, simplifying the extraction of embedded plans. The adoption of more complex workflows, containing cycles, and the extraction of corresponding embedded plans might be useful and are still under investigation.

6 Related work

Our strategy may be compared to planning techniques adopted in several application domains, such as Web service composition [16–19]. Dan Wu et al. [20] use SHOP2, a hierarchical task network (HTN) planner, to automatically compose Web services described in DAML-S (now OWL-S). Sheshagiri et al. [21] adopt a backward-chaining algorithm for Web service composition, where DAML-S service descriptions are translated into a set of STRIPS-like planning operators by a set of rules.

TALPLANNER [22], inspired by the fast forward-chaining planner TPLAN of Bacchus and Kabanza [23], uses a temporal action logic which enables it to plan for incompletely specified dynamic worlds. The ARPA/Rome Laboratory Planning Initiative (ARPI) also promotes the implementation of innovative planners [24].

The composition of programs and tools also bears some similarity with our approach. As an example, the electronic tool integration platform (ETI) [25] is an environment where a user may combine functionalities of several tools to obtain sequential programs, called coordination sequences. ETI uses concepts from temporal logic to compose the desired sequences. This platform has been redesigned to explore the standard Web service technology [19], offering new service composition features.

The flexible architecture we envision for our project resembles the modular characteristics of O-Plan (the open planning architecture) [26,27] elaborated by the AI—Artificial Intelligence Applications Institute at the University of Edinburgh. This AI based planner has evolved as a sort of a planning service plug-in, and it is now available as a component of its successor, the I–X integration platform [28].

SIPE-2 interactive planner [29] has a long research history and has been tested in real-world situations. Among other applications, it was used in an oil spill response system. Therefore, in this aspect, SIPE-2 is the closest initiative related to our intended application domain.

7 Conclusions and future work

In this paper, we introduced a mechanism that helps reduce workflow complexity and adds flexibility to workflows through incremental planning. The proposal uses a planner component that features a library of predefined workflows. Our approach is motivated by the design of emergency response systems that account for real-life accidental scenarios. We are currently designing a new prototype version of the INFOPAE system [30], an emergency response system in use in a large number of installations.

A fundamental requirement for effective emergency response, whether involving the rescue of stranded astronauts, evacuation of civilians, or recovery from a natural disaster, etc., is to be able to determine, given any predictable emergency situation, at least one plan that will bring the system back to a desired state. The plan must be flexible enough to account for a series of in-between difficulties, such as resource unavailability or lack of information.

In this paper, we proposed a strategy to simplify the construction of emergency response plans by combining a library of pre-defined (simple) workflows with a planner that uses intermediate goals, generated by condition goal inference rules. Such rules indeed enable the planner to generate plans in an incremental fashion.

We are currently evaluating questions related to the completeness of the proposed strategy. A completeness criterion would be to require that the planner generates (correct) plans for all possible emergency scenarios. This criterion is not reasonable, however, given the large number of emergency scenarios that would result from any reasonably complex industrial installation and the low probability associated with some of the more complicated emergency scenarios (e.g., all tanks in a refinery fortuitously catching fire at the same time). Instead, we are considering a weaker completeness criterion, in which we would require that the planner generates (correct) plans for all possible emergency scenarios for each individual components of the complete installation, and then postulate that the individual plans can be combined as necessary.

Acknowledgments This work is partially supported by CNPq under grants 550250/2005-0, 551241/2005-5 and 552068/02-0.

References

1. Bruegge B, O'Toole K, Rothenberger D (1994) Design considerations for an accident management system. In: Proc. conf. cooperative information sys., pp 90–100
2. Van de Walle B, Turoff M (2007) Emergency response information systems: emerging trends and technologies special section—introduction. Commun ACM 50(3):29–31
3. Van de Walle B, Turoff M (2007) Decision support for emergency situations. In: Burstein F, Holsapple C (eds) Handbook on Decision Support Systems, Internat. Handbook on information systems series. Springer, Heidelberg
4. Casanova MA, Coelho TAS, Carvalho MTM, Corseuil ELT, Nóbrega H, Dias FM, Levy CH (2002) The Design of XPAE—

an emergency plan definition language. In: IV Simpósio Brasileiro de GeoInformática - Caxambú, MG, Brasil, 2002
5. Carvalho MT, Freire J, Casanova MA (2001) The architecture of an emergency plan deployment system. In: III Workshop Brasileiro de GeoInformática, Instituto Militar de Engenharia, Rio de Janeiro, Brasil, October 2001, pp 19–26
6. Ciarlini AEM, Veloso PAS, Furtado AL (2000) A formal framework for modelling at the behavioural level. In: Proceedings of the European–Japanese conference on information modeling and knowledge bases, pp 67–81
7. van der Aalst WMP, Barros AP, Hofstede AHM, Kiepuszewski B (2000) Advanced workflow patterns. In: Conf. on coop. information systems, pp 18–29
8. Furtado AL, Ciarlini AEM (1997) Plots of narratives over temporal databases. In: Proceedings of 8th international workshop on database and expert systems applications (DEXA '97)
9. Ciarlini AEM, Furtado AL (1999) Simulating the interaction of database agents. In: Proceedings of DEXA'99 database and expert systems applications conference, Florence, Sept. 1999
10. Ciarlini AEM, Pozzer CT, Furtado AL, Feijó B A (2005) A logic-based tool for interactive generation and dramatization of stories. In: Proc. ACM SIGCHI international conference on advances in computer entertainment technology (ACE 2005), Valencia, Spain
11. Yang Q, Tenenberg J, Woods S (1996) On the implementation and evaluation of abtweak. Comput Intell J 12(2):295–318
12. Chapman D (1987) Planning for conjunctive goals. Artif Intell 32:333–377
13. Reiter R (1978) On Closed World Databases. In: Gallaire H, Minker J (eds) Logic and databases. Plenum, New York pp 55–76
14. Schank RC, Abelson RP (1977) Scripts, plans, goals and understanding. Erlbaum, Hillsdale
15. Furtado AL, Ciarlini AEM (2001) Constructing Libraries of Typical Plans. In: Proceedings of the 13th international Conference on advanced information systems engineering, 2001
16. Srivastava B, Koehler J (2004) Planning with workflows—an emerging paradigm for web service composition. In: ICAPS 2004 Workshop on planning and scheduling for web and grid services, Whistler, British Columbia, Canada, June 2004
17. Martínez E, Lespérance Y (2004) Web service composition as a planning task: experiments using knowledge-based planning. In: ICAPS 2004 workshop on planning and scheduling for web and grid services, Whistler, June 2004
18. Carman M, Serafini L, Traverso P (2003) Web service composition as planning. In: ICAPS 2003 Workshop on planning for web services, Trento, June 2003
19. Margaria T (2005) Web services-based tool-integration in the ETI platform. SoSyM Int J Softw Syst Modell 4(2): 141–156
20. Wu D, Parsia B, Sirin E, Hendler J, Nau D (2003) Automating DAML-S web services composition using SHOP2. In: Proceedings of 2nd Int'l. SemanticWeb Conf. (ISWC2003), Florida, 2003
21. Sheshagiri M, desJardins M, Finin T (2003) A planner for composing services described in DAML-S. In: ICAPS 2003Workshop on Planning forWeb Services, Trento
22. Doherty P, Kvarnström J (2001) TALplanner: a temporal logic based planner. AI Magazine 22(1):95–102
23. Bacchus F, Kabanza F (2000) Using temporal logics to express search control knowledge for planning. Artif Intell 116(1–2):123–191
24. Tate A (1996) Advanced planning technology: technological achievements of the ARPA/Rome laboratory planning initiative. AAAI press, Menlo Park
25. Steffen B, Margaria T, Braun V (1997) The electronic tool integration platform: concepts and design, special section on the electronic tool integration platform. Int J Softw Tools Technol Transfer 1:9–30
26. Tate A, Drabble B, Dalton J (1994) Reasoning with constraints within O-Plan2. In: Burstein M (ed) Proceedings of the ARPA/Rome laboratory planning initiative workshop, Tucson. Morgan Kaufmann, Palo Alto
27. Tate A, Dalton J, Levine J (1999) Multi-perspective planning—using domain constraints to support the coordinated development of plans. O-Plan Final Technical Report AFRL-IF-RS-TR-1999–60, April 1999
28. Tate A (2000) Intelligible AI planning—generating plans represented as a set of constraints. In: Research and development in intelligent systems XVII, Proceedings of ES2000, the twentieth British Computer Society Special Group on expert systems international conference on knowledge based systems and applied artificial intelligence, Cambridge. Springer, Heidelberg, pp 3–16
29. Wilkins DE (1998) Practical planning: extending the classical AI planning paradigm. Morgan Kaufmann, San Mateo
30. Carvalho MT, Casanova MA, Torres F, Santos A (2002) INFOPAE—an emergency plan, deployment system. In: Proceedings of the international pipeline conference, Calgary