A Framework for Dynamic Composition and Management of Emergency Response Processes

Abeer Elahraf, Ayesha Afzal, Ahmed Akhtar,
Basit Shafiq, Jaideep Vaidya, Shafay Shamail and Nabil R. Adam

Abstract—An emergency response process outlines the workflow of different activities that need to be performed in response to an emergency. Effective emergency response requires communication and coordination with the operational systems belonging to different collaborating organizations. Therefore, it is necessary to establish information sharing and system-level interoperability among the diverse operational systems. Unlike typical e-government processes that are well structured and have a well-defined outcome, emergency response processes are knowledge-centric and their workflow structure and execution may evolve as the incident unfolds. It is impractical to define static plans and response process workflows for every possible situation. Instead, a dynamic response should be adaptable to the changing situation. We present an integrated approach that facilitates the dynamic composition of an executable response process. The proposed approach employs ontology-based reasoning to determine the default actions and resource requirements for the given incident and to identify relevant response organizations based on their jurisdictional and mutual aid agreement rules. The Web service APIs of the identified response organizations are then used to generate an executable response process that evolves dynamically. The proposed approach is implemented and experimentally validated using an example scenario derived from the FEMA Hazardous Materials Tabletop Exercises Manual.

Index Terms—Business process composition, Emergency response processes, Knowledge-centric business processes, process evolution.

1 Introduction

An emergency response process is a workflow of different activities that need to be performed in response to an emergency situation. Effective emergency response planning requires communication and coordination with the diverse operational systems belonging to different collaborating organizations, including government agencies, non-government organizations (NGOs), and private sector entities. Therefore, a major requirement for the development of an emergency response process is to establish information sharing and system-level interoperability among the operational systems of collaborating organizations. Unlike the traditional business processes (e.g., e-government and e-commerce processes) which have well-defined and predictable workflow structure and outcomes, emergency response processes are knowledge-driven and are executed in a dynamic and non-deterministic environment. Their workflow structure emerges and evolves dynamically depending upon the environmental context, user decisions, availability of resources, and the rules and policies of response organizations. Establishing interoperability is particularly challenging in such an environment, as not only the workflow structure but also the collaborating organizations, data sources, and resource providers may not be known a priori and may need to be discovered at runtime. Moreover, there can be new organizations that may not have collaborated earlier. Therefore, we need an approach to enable on-the-fly composition of emergency response processes by enabling information sharing and interoperability among the information systems of relevant response organizations.

Service-oriented architecture provides the key enabling technology to establish interoperability and coordination among the applications and systems of response organizations for the composition and management of emergency response processes. There are some service-oriented platforms and systems for interoperable information exchange for emergency management and response planning including, XchangeCore[1], formerly known as Unified Incident Command Decision Support System (UICDS) [1], [2], FEMA Incident Resource Inventory System (IRIS) [3], Service-Oriented ArchiteCtures Supporting Networks of Public Security (SoKNOS) [4], and social media alert and response to threats to citizens (Smart-C) [5]. Table 1 provides a comparison of these platforms/systems with respect to their capabilities to support dynamic composition of emergency response processes. Specifically, we compare the capabilities of these platforms/systems in terms of: (i) information sharing among diverse systems; (ii) reasoning support for response planning; (iii) dynamic discovery of response organizations; and (iv) automated process composition by interfacing with the diverse operational systems of response organizations. Although these systems employ semantic-based reasoning for emergency response planning and decision support, they do not support on-the-fly com-

- *A. Elahraf, J.Vaidya, and N.Adam are with Rutgers University, Newark, NJ 07102. E-mail: aelahraf@newark.rutgers.edu, jsvaidya@rbs.rutgers.edu, adam@adam.rutgers.edu.*
- *A. Afzal is with the Department of Computer Science, Air University, Pakistan. E-mail: ayesha@aumc.edu.pk*
- *A. Akhtar, B. Shafiq and S. Shamail are with the Department of Computer Science, Lahore University of Management Sciences, Lahore, Pakistan.E-mail: 16030059@lums.edu.pk, basit@lums.edu.pk, sshamail@lums.edu.pk*

1. https://www.saberspace.org/xchangecore-home.html

TABLE 1: Comparison of Interoperable systems for emergency management and response planning

	Information sharing	Reasoning support	Dynamic discovery of response organizations	Automated process composition
UICDS/XchangeCore	Yes	No	No	No
IRIS	Yes	Yes	No	No
SoKNOS	Yes	Yes	Yes	No
Smart-C	Yes	Yes	Yes	No
Proposed Approach	Yes	Yes	Yes	Yes

position of response processes that may evolve based on the situation at-hand.

Fig. 1 shows the composition environment for emergency response process management. The response organizations provide access to their operational systems through Web services and publish the corresponding application programming interfaces (APIs). The response process is composed by integrating the diverse systems and Web service APIs of response organizations, as depicted in Fig. 1. The response process may need to be updated and recomposed with the addition of new activities due to evolving situation and new requirements. This requires evaluating the latest situation, identifying the agencies and organizations that can provide the required resources, and integrating their Web services into the instantiated response process.

The following example scenario illustrates the need for supporting such a dynamic composition of the response process.

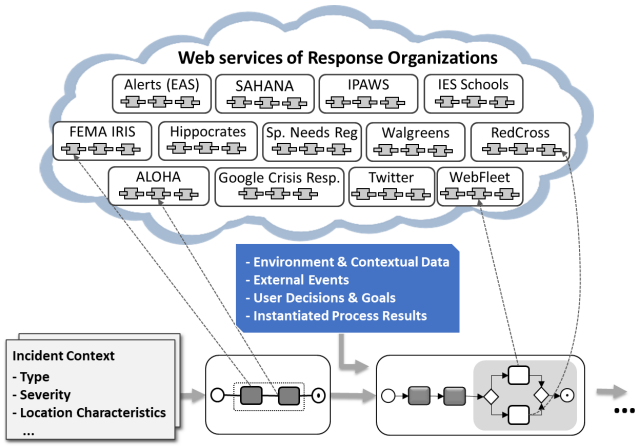


Fig. 1: Composition environment for emergency response process management.

Rail incident scenario: Consider a rail incident scenario similar to FEMA Hazardous Materials Tabletop Exercises Manual Rail Incident Scenario [6], in which a freight train is reported to collide with a car at a crossing over the tracks. First responders approach the area to find a large greenish smoke cloud emanating from the wreckage. An initial incident command structure is established, and the incident commander (IC) determines the following response actions:

- Request dispatch of initial response resources locally from the city (including EMS, law enforcement, fire and HazMat teams).
- Determine the risks posed by the hazards and identify the threat zones and safe distances.
- Order evacuation of the hazard area as per the identified threat zones.
- Notify appropriate regulatory agencies, schools, and the general public within the area through Emergency Alert System (EAS) about the evacuation and sheltering information.
- Activate other public warning systems (e.g., social media and radio alerts, etc.).
- Identify local resources that can be committed and are ready to dispatch.

The situation further evolves when the city Dispatch Center informs the IC of the expected arrival of a northbound passenger train in half an hour. Responders assisting with the evacuation also report that numerous people need specialized transportation (i.e., wheelchair-bound, on oxygen, bedridden, hearing, and vision impaired). Based on the current situation, IC decides to take the following actions:

- Evaluate the need for additional resources to deal with traffic control (portable road signs, redirect traffic flow patterns, and limit access).
- Instruct the railroad company to take steps for halting all incoming train traffic.
- Request resources to assist with special needs evacuation.
- Request additional resources from other states to assist with the evacuation (transportation, food, and medical supplies).

Similarly, the incident commander may take additional actions as the situation evolves. These actions correspond to extending the instantiated response process with more activities.

Currently there is no unified system that supports automatic interfacing with the operational systems of response organizations for response process composition and management. Moreover, existing approaches for adaptive process composition [7], [8], [9] are limited to structured BPs, which employ event-condition-action rule based reasoning for process adaptation. The rules are defined for different type of events e.g., replacing a failed service by a pre-defined service. However, emergency response process management involves certain types of events for which the concrete actions may not be known a priori. For example, if a given resource request cannot be satisfied by a local agency, then the request is sent to a state-level agency. However, the specific state-level agencies and their service APIs need to be determined at runtime. Such events trigger

re-composition of the emergency response process which is the main focus of this paper. Our key contribution is to ensure that the needed process recomposition can be automatically and dynamically performed as the situation evolves.

Our proposed approach provides an integrated framework for dynamic composition of an executable response process. It employs ontology-based reasoning to determine the default actions and resource requirements for the given incident and to identify relevant response organizations and APIs of their operational systems based on their jurisdictional and mutual aid agreement rules. The discovered Web service APIs of the different response organizations are then used to generate an executable response process that evolves dynamically based on any changes in the environmental context as well as the availability of resources. We experimentally validate the effectiveness of the proposed approach using an example scenario derived from FEMA Hazardous Materials Tabletop Exercises Manual.

The remainder of this article is organized as follows. Section 2 presents a review of related works. Section 3 provides a formal statement of the problem. Section 4 presents the proposed approach. Section 5 presents the results of the experimental evaluation. Section 6 presents a prototype implementation of the proposed framework. Finally, Section 7 concludes the article and discusses future work directions.

2 Related Work

Automated business process composition and management has been extensively studied in the literature. Significant work has been carried out on dynamic service composition, process adaptation and schema evolution in the context of structured business processes. However, there is a lack of integrated approaches that address the dynamic composition requirements for knowledge-driven processes that are unpredictable and emergent, as discussed in the Introduction. Below we discuss the key requirements [8], [10] and related approaches for dynamic process composition. For completeness, we discuss the work that tackles the issues of schema evolution, runtime process adaptation, and knowledge-driven business process development. However, our work primarily tackles the issues in knowledge-driven business process development. We do not handle schema evolution, or service failure/replacement issues since these can be directly handled using prior work.

Schema evolution requirements mostly arise due to process re-engineering efforts, e.g., due to changes in business rules, policies, regulation, business strategy, and continuous process improvement. Such changes in schema typically require developing a new version of process schema/model, which, in turn affects the running instances of the process as well [8]. Schema change problem has been extensively studied, and several approaches have been proposed. These include graph based model matching and merging techniques [11], [12], [13], [14], as well as rule-based techniques [7], [15], [16], [17]. Graph-based techniques support flexibility in process models to enable process designers to compare and reuse configurable elements [11] in existing process models. These approaches mainly address design time variability in process workflows. Rule based approaches mostly support dynamic changes in process schema by enabling different kinds of ad-hoc deviations from a given process model.

Runtime process adaptation requirements mostly arise due to abnormal termination or failure of tasks (e.g., due to underlying web service failure or unavailability), and/or violation of any constraints related to data (e.g., missing input), tasks (e.g., pre/post-condition not satisfied), and/or temporal requirements (expiration of a deadline), etc. [8]. These requirements have been addressed by flexible and adaptive service composition approaches that mostly rely on exception handling mechanisms with the associated recovery procedures built manually by a process designer at run-time. ADEPT2 [15], for example, supports handling of unanticipated exceptions by enabling different kinds of ad-hoc deviations from a given process model.MoDAR [7] is a rule-based model-driven development approach for adaptive service composition. Similarly, Sabatucci et al. [9] proposed a rule-based approach employing static analysis of the global workflow state to support adaptive workflows.

We consider process adaptation only in the sense of dynamic recomposition of emergency response processes in order to respond to changes in the environment or emergence of new requirements. In this paper, we do not focus on the issue of service replacement/failure as well as schema changes and assume that the existing approaches discussed above can be employed for handling such issues. However, from an implementation perspective, we also utilize a rule-based adaptive process composition approach to handle such issues.

Knowledge-driven business process development involves taking into consideration three main characteristics of knowledge-driven processes which differentiate them from structured business processes; (i) Unpredictable and emergent process workflow structure - The activities and process structure are not predefined and is determined dynamically based on the knowledge of environmental context, situation, and case-specific parameters that are not known a priori and are subject to change at runtime; (ii) Goal-oriented composition - Composition goals may change, and new goals may arise as new data or actions emerge during process instantiation and execution; and (iii) Constraint & rule driven composition - policies, rules, and regulations can influence the process structure and constrain its execution. While there is a lack of integrated approaches addressing these requirements, some works address different requirements in knowledge-driven business process development in a piecemeal manner.

Marella et al. proposed the SmartPM framework [16], which builds on Knowledge Representation and Reasoning techniques using situation calculus, logic-based programming, and planning. SmartPM framework supports monitoring and run-time adaptation of existing compositions and does not include support for automated service composition and process evolution. Its current implementation relies on a process designer for specification of the process participants and model. However, an automated composition approach can be integrated with SmartPM to replace the BPMN-based manual process design approach.

In another work, Bucchiarone et al. [18] proposed a goal-oriented business process adaptation approach based on context evolution. At a high level, the proposed approach verifies the evolution of the process context and process execution against a desired model that encodes the business policies over some domain elements. Upon detection of any deviation, the adaptation process is triggered. Adaptation involves automatic service composition and execution, considering the current state of the process and its context. In a recent work, Bucchiarone et al. proposed another framework for context-aware dynamic composition of process fragments [19], [20]. In this work, reusable process fragments [11] are considered as composable components [21] to support the composition of adaptive-by-design business processes. The underlying idea is that different entities (service providers) in the system publish their functions through process fragments that can be used in new process compositions as well as for dynamic process evolution. The approach is quite similar to our work, but it does not consider the problem of search and selection of existing fragments related to required service composition. Moreover, with respect to execution support, in these existing frameworks, the composition logic can only be interpreted and executed by the internal execution engines developed for the specific platforms. On the contrary, our proposed approach provides an integrated framework for composition and executable code generation for knowledge-driven business processes. The executable process can be deployed on any process execution engine.

3 Problem Statement

We address the problem of automated composition and management of emergency response process considering a dynamic environment. We formalize this problem as follows:

Given:

i. *the contextual information of the incident including incident type, severity, and location;*
ii. *the default actions or static response plans pre-defined for specific emergency events;*
iii. *response organizations, their jurisdictional and mutual aid agreement rules and the resources they can provide;*
iv. *available Web services/ APIs of the operational systems of response organizations to request their resources;*

compose an executable response process that can be instantiated for the incident at-hand, and that can adapt and evolve as the incident unfolds or as the environmental context changes.

Here an executable response process refers to a business process in which each activity is bound to the appropriate operational system Web service/ API and is deployed on a process engine for instantiation. We assume that the functionality of the underlying operational systems of response agencies and resource provider organizations is exposed through their respective Web service operations. Therefore, each activity in the abstract workflows of default actions is realized through a composition of Web service operations of the potential response organizations.

4 Proposed Approach

The proposed approach for dynamic composition of an emergency response process is a multi-step approach that incrementally generates an executable response process and enables adaptability to a changing situation as the incident evolves. Fig. 2 depicts the architectural view of our proposed framework. This framework employs ontology and reasoning engine for response planning for a given incident. The ontology characterizes incidents based on type, severity and location; the default actions and resources required for resolving different incident types; and the response agencies and organizations.

Based on the given contextual information of the incident at hand, the *reasoning engine* searches the ontology and retrieves the default actions, required resources, and the response agencies responsible for performing these actions and providing the resources. The default actions are encoded in the ontology as *abstract process fragments* which are essentially workflows of activities that need to be performed in response to specific situations. For example, in case of a train crash, the default actions include dispatching EMS responders to the crash site, obtaining the train consists (i.e., the train and wagon numbers, and the type and quantity of material in containers) from the railroad company, finding the hazard classes and risks, establishing hazard control zones and requesting transportation resources for evacuation, etc. [6]. In addition, the reasoning engine also discovers and selects the APIs of the operational systems/Web services of the relevant response agencies and resource providers based on their jurisdictions, rules, policies.

Given the default actions and the operational systems/ service APIs of the resource provider systems identified by the reasoning engine, the *process composition modeling* component generates an integrated response process that enables the incident commander to request resources from the selected resource provider agencies by interacting with their appropriate operational systems. Essentially, this component determines an execution order of activities in the default actions and binds each activity to appropriate operational systems and services of the response agencies. We employ a reachability analysis based service composition approach for the generation of such an executable response process. This executable response process is then presented to the user for any customizations (e.g., adding/removing some activities). *Code generation component* of the system then generates executable process code (e.g., in BPEL language) for deployment and instantiation on a process execution engine.

We utilize a rule-based adaptive process composition approach to handle service replacement/failure issues as well as schema changes [22]. Essentially, our process composition approach is an event-condition-action rules based approach considering different types of events including, (i) service failure/unavailability; (ii) error messages returned by a service (e.g., due to service interface changes, input validation error) (iii) change in process execution status (e.g., updated availability status of requested resources, updated threat zone); and (iv) user actions (e.g., addition/removal of a response activity, making resource re-

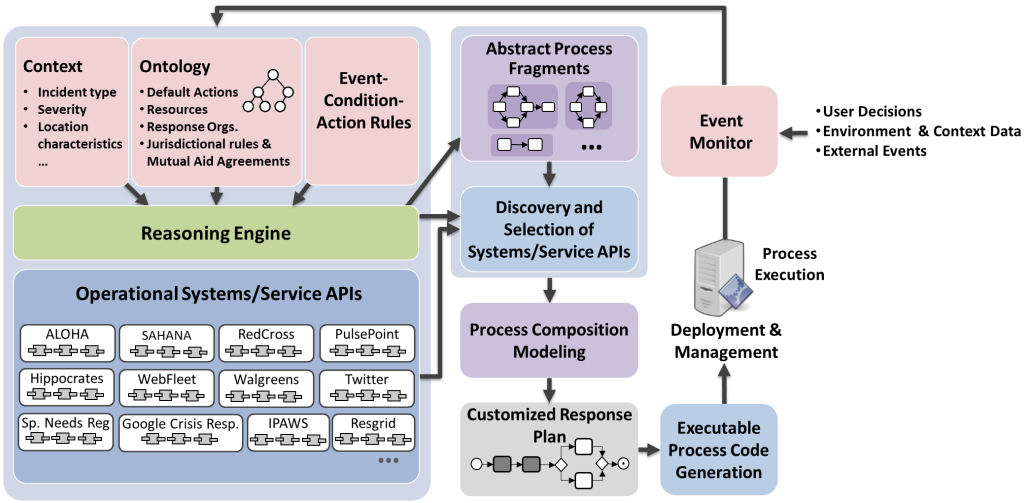


Fig. 2: Architectural view of proposed framework for dynamic composition & management of emergency response processes.

quest to a specific response organization). Similar to existing works [16], [22], our approach includes a rule base and a rule engine. The rule base includes event-condition-action rules defining the actions for each event type. For example, failure or unavailability of a service will trigger a service replacement action. Similarly, rules can be defined for compensation actions in case of service failures. For example, roll-back or cancellation of previously executed service(s) by invoking the appropriate service API. In addition, there are certain types of events, for which the concrete action may not be known a priori. For such events, our proposed approach enables extension and recomposition of an instantiated response process based on the changes in environmental context (e.g., a chemical leakage incident evolves into a fire incident) or emergence of new requirements (e.g., more fire trucks are needed, etc.). As the incident evolves, the proposed approach extends and recomposes the instantiated process in an iterative manner until the incident is resolved.

In the following subsections, we discuss each component of the proposed system in detail.

4.1 Emergency Management Ontology

A key component of our proposed system is the emergency management ontology, which describes the essential concepts and their relationships in the emergency management domain. The key concept in the ontology is the '*incident*' class, which describes an emergency situation in terms of the incident type, location, severity, and potential hazards etc. An incident is associated with an '*incident-type*' class which describes different types of incidents e.g., a railroad accident, a chemical plant fire, hurricane or earthquake. Each *incident-type* is associated with: (i) 'default actions'; (ii) required 'resource types'; and (iii) response organizations.

Default actions are abstract process fragments modeling high-level workflow of response activities. We represent the default actions in the standard Business Process Model and Notation (BPMN). Fig. 3 shows an example abstract process fragment of default actions for a HazMat train accident. Note that the activities in default actions are only defined at an abstract level and lack any information about the specific operational systems for communication and coordination with the response organizations.

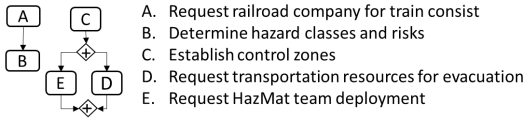


Fig. 3: Default actions for hazardous materials carrying train accident.

Emergency Management Resources: Emergency management resources are described in the ontology based on NIMS standard resource types. Each *resource* has an *owner* agency/ organization that operates in a given jurisdiction and can potentially provide the resource for assignment to an incident e.g., a firefighting helicopter resource owned by a county fire department.

Response organizations: A response *organization* represents a government agency, non-governmental organization, or a private entity that is directly or indirectly involved in emergency management activities. An organization has properties including, name, roles, jurisdiction (city, town, county, state, or federal), location, rules/policies, the type of resources it can provide, as well as the links to their operational systems/ Web services APIs (WSDL files). These APIs can be used to query the internal databases and operational systems of the response organizations for resource availability and commitment.

Rules: The emergency management ontology also encodes the rules of response organizations for responding to emergencies. These rules can generally be categorized into the following two types:

- *Jurisdictional Rules* – These rules specify the jurisdictions and responsibilities of the response organizations. For example, consider the following rules for determining relevant government agencies for the response process:

  - Local agencies must respond to a disaster within their jurisdiction
  - If local agencies have their resources exhausted, then the request should be escalated to county, state and federal level agencies in the same order.
- *Mutual Aid Agreement Rules* – All emergencies originate at the local level; however, they can escalate, warranting the need for mutual aid from outside of the affected county and contiguous counties. Mutual aid agreement rules specify the agreements between agencies for collaboration with other public and private agencies operating in the same or different jurisdiction. As an example, consider a county-level mutual aid agreement rule which states that the Bergen county fire department in New Jersey state will support firefighting operations in a neighboring county in the New York state when requested.
  There are several types of mutual aid agreements including basic contracts between government/private organizations, local, regional, inter-state, intra-state, and international agreements for assistance in the form of personnel, equipment, materials, and other associated services [23].

We encode the jurisdictional and mutual aid agreement rules of response agencies in the ontology using the Semantic Web Rule Language (SWRL). These rules are considered for identifying the response agencies, as discussed in the following section.

4.2 Reasoning engine

The ontology reasoning engine is one of the core components of the system. Given the incident's contextual properties including incident type, severity and location, the reasoning engine performs reasoning on the incident ontology to determine the default actions, resources required, and the response organizations based on their jurisdictions and mutual aid agreement. As discussed in section 4.1, the default actions are only abstract process fragments that define a set of activities that need to be performed for a particular incident type. Each activity in the default action is realized through a composition of multiple Web services that expose the functionality of the operational systems of the response agencies and resource provider organizations. Reasoning engine is responsible for the discovery and selection of relevant APIs of the response organization systems, for the realization of activities in the default actions. This step involves taking into account the jurisdictional rules and mutual aid agreements between organizations.

Given the default actions and APIs of the identified response organizations' systems, the next step is to compose a response process that enables the incident commander to request resources from the response agencies by interacting with their appropriate operational systems. We discuss our approach for this step in the following subsection.

4.3 Process composition modeling

Process composition modeling involves elaborating the default actions identified for the situation at hand into a concrete response process. As discussed earlier, the default actions are only abstract workflows of response activities, which lack information about their execution order and the specific operational systems of response organizations for execution of the underlying response activities. In order to compose a concrete response process, we employ a reachability analysis based approach that determines the execution order of the activities in the default actions, and bind these activities to appropriate operational systems and service APIs.

Below we provide the important definitions followed by a detailed description of the approach.

Definition 1. *(Web service operation):* A Web service operation is defined as a 5-tuple [24],

$$s = (\textit{op-name}, A_{in}, A_{out}, C_{pre}, C_{post}), \text{ where}$$

- *op-name* corresponds to the name of the Web service operation;
- A_{in} is the set of input parameters/ attributes;
- A_{out} is the set of output parameters/ attributes;
- C_{pre} represents *preconditions* of s, defined with respect to the values of some attributes *before execution of* s;
- C_{post} represents *postconditions* of s, defined with respect to the values of some attributes *after execution of* s.

Our proposed approach builds on the Colored Petri net (CPN) reachability analysis based service composition approach [25]. The idea is to explore those paths in the CPN reachability tree that satisfy a particular goal state of the system given some initial state. Such paths denote the possible execution orders of Web services for response process composition.

Definition 2. *(Colored Petri net)*: A colored Petri net [25] is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$, where:

- Σ is a finite set of non-empty types, called color sets.
- P is a set of places.
- T is a set of transitions.
- A is a set of arcs, such that $P \cap T = P \cap A = T \cap A = \emptyset$. The arcs are categorized as normal arcs, read arcs, and inhibitor arcs.
- N is a node function, $N : A \rightarrow P \times T \cup T \times P$.
- C is a color function $C : P \rightarrow \Sigma$
- G is a guard function defined from T into expressions such that , $\forall t \in T : [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \Sigma]$.
- E is an arc expression function defined from A into expressions such that , $\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$.
- I is an initialization function defined from P into closed expressions such that , $\forall p \in P : [Type(I(p)) = C(p)_{MS}]$.

A distribution of tokens on the places is called a *Marking*. An *initial marking* is determined by evaluating the initialization expressions: $\forall (p, c) : initMrkg(p, c) = (I(p))(c)$, where (p, c) denotes a token element, $p \in P$ and $c \in C(p)$.

In the response process composition context, we model the set of selected service operations as a set of transitions T in the CPN model. A transition is linked to its corresponding input and output places. Each place can hold certain colored tokens. A colored token corresponds to the object class associated with the input or output

parameter of a service operation. The attributes of a colored token correspond to the attributes of an object class. The preconditions of a service operation are specified as guard function and the postcondition as an arc expression to the output place(s) in the CPN model. A transition can fire if its input places have the required tokens and its guard function evaluates true. Firing of a transition implies invocation of the corresponding service operation when all its input parameters are available, and its preconditions are satisfied.

Example 1 illustrates our process composition approach.

Contextual Attributes	Initial State
incident_type = `HazMat Train Accident'; severity =`Level 1'; location=`Urban'; city = `Saddle Brook'; state = `NJ'; latitude = '40.904009'; longitude = '74.100732'; train_id = `RAIX1102'; ...	incident_type: `not-null'; severity: `not-null'; location: `not-null'; city: `not-null'; state: `not-null'; latitude: 'not-null'; longitude: 'not-null'; train_id: `not-null'; ...

Fig. 4: Example: Computing initial state from contextual attributes

Example 1. For the train incident scenario discussed in Section 1, we first determine an initial state of the system by considering the contextual attributes of the incident, as shown in Fig. 4.

In order to compute a goal state for process composition, we consider the system state after successful execution of activities in the retrieved default actions. For example, consider activities A, B, and C in the default actions of Fig. 3 for finding train consists, hazard classes & risk, and establishing control zones, respectively. Each of these activities has associated input and output attributes. For example, activity A requires $train\_id$ as input and returns $materials$ as output, B requires $materials$ as input and returns $hazardClass$ and $risks$ as output, similarly C requires $materials$, $latitude$ and $longitute$ as input and returns $safeDistances$ and $controlZones$ as output. The combined effect of the output attributes of these activities determine the goal state of the default action as shown Fig. 5, where the output attributes $material$, $hazardClass$, $safeDistances$, and $controlZones$ are assigned a "not-null" value in the goal state.

Default Actions	Goal State
A. Request railroad company for train consists	materials: `not-null';
B. Determine hazard classes and risks	hazardClass: `not-null'; risks: `not-null'; safeDistances:`not-null';
C. Establish control zones	controlZones: `not-null';

Fig. 5: Example: Computing goal state from activities in the default actions.

Fig. 6 illustrates our CPN reachability analysis based service composition approach. Given the initial state, goal state, and the CPN models of the selected Web service operations, state-space reachability analysis is performed to determine an execution order of services that satisfies the composition goal. Fig. 6(a) shows the CPN structures, including the places corresponding to the input/output object classes, and the transitions corresponding to the selected service operations. As shown in the figure, firing of transition t_0, which does not have any guard condition places the tokens in the initial state in their corresponding places $Train$ and $Location$. Once the token $train\_id$ becomes available, transition t_1's guard condition is satisfied and it is fired. Thus, tokens $material$ and $quantity$ are added to the place $Train\_Consists$. The transition firing process continues until the placement of tokens (current marking) satisfies the goal state, or there are no more transitions that can be fired. Fig. 6(b) shows the corresponding transition firing sequence for the CPN structures given in Fig. 6(a).

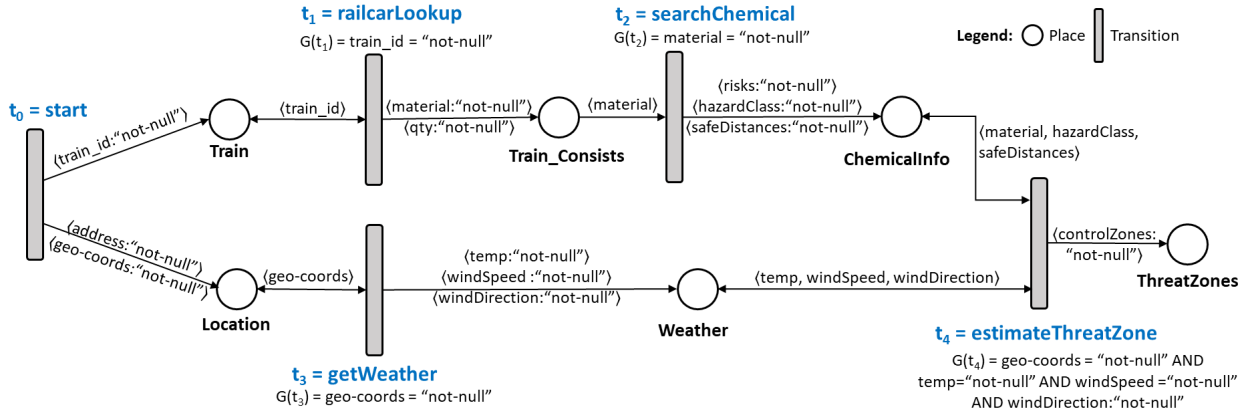
Note that we assume that there is no syntactic and semantic heterogeneity between the attributes of the response activities in the default actions, and the input/output parameters of service operations of response organizations i.e., these attributes are specified using the same terms in the default actions as well as in the appropriate Web service operations. If heterogeneity does actually exist between attribute names, we can employ the existing attribute-based matching approaches [24], [26] to resolve differences in attribute names before continuing with the reachability analysis.

Given the default actions determined by the ontology reasoning component, Algorithm 1 computes the workflow of the concrete response process. The algorithm takes as input, a default action $\mathcal{F}$, CPN model of the selected Web services $\in S$, and the set $C = \{\langle a_1, v_1\rangle, \ldots, \langle a_k, v_k\rangle\}$, containing the contextual attributes (a_i) with their respective values (v_i). The output is a composition of services that satisfies the goal state. The algorithm first computes the initial and goal states for required service composition in S. *computeInitialState()* function computes the state of the system from contextual attributes in C as follows:

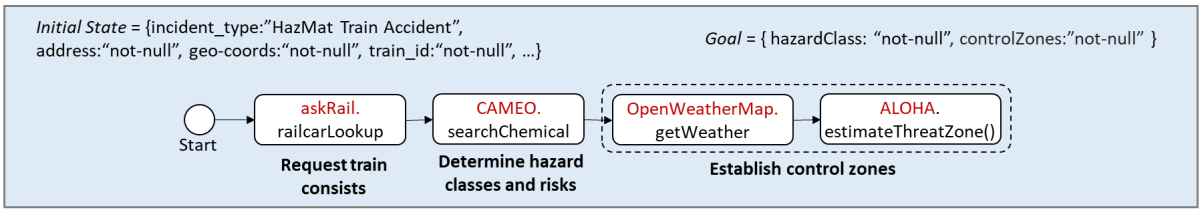
a. All the attributes whose values are given in C are assigned abstract values, "null"/ "not-null" or original values in case of enumerated data types.
b. All the attributes whose values are not-known are assigned "X".
c. Relevant tokens are identified and inserted in their respective places with their values - This placement of tokens corresponds to the input marking or initial state.

In the next step, the *computeGoalState()* function computes the expected state of the system after successful execution of activities in the given default action, $\mathcal{F}$ as follows:

a. Find the post-condition attributes of each default action $\in \mathcal{F}$ by considering the expected output of component activities.
b. Assign "null", "not-null" or original value to the identified attributes according to the post-conditions of their respective activities.
c. Identify relevant tokens and insert in their respective places with the computed attribute values - This placement of tokens corresponds to the output marking or goal state.



(a) CPN structures showing placement of tokens in places and enabled transitions



(b) Initial and Goal states with corresponding transition firing sequence in the CPN reachability tree

Fig. 6: Response Process Composition

Given the initial state, goal state, and the CPN model, the reachability analysis algorithm (Algorithm 2) is then invoked to compute a possible composition of selected Web services (line 3). If reachability analysis algorithm returns a non-empty sequence of transitions, then the corresponding sequence of services is considered a valid service composition workflow R for the given default action.

ALGORITHM 1: FindServiceComposition

Input: $\mathcal{F}$: A default action
Input: CPN: Colored Petri net structures of available service operations $\in S$
Input: $C = \{\langle a_1, v_1\rangle, \ldots, \langle a_k, v_k\rangle\}$: The set of contextual attributes (a_i) with their respective values (v_i).
Output: R: Workflow of the concrete response process
1: $Init \leftarrow$ computeInitialState(C)
2: $Goal \leftarrow$ computeGoalState($\mathcal{F}$)
3: $R \leftarrow$ ReachabilityAnalysis($Init, Goal, CPN$)
4: **return** R

Algorithm 2 provides the pseudo-code of our CPN reachability analysis algorithm, which uses the occurrence graph method proposed in [25]. Given an initial state $Init$, goal state $Goal$, and the CPN model of the available service operations, Algorithm 2 performs the reachability analysis by constructing the CPN occurrence graph to find a service composition. The returned service composition is represented as a sequence of transitions fired to reach from the initial state to the goal state. If no such path exists, the algorithm returns an empty set. The algorithm maintains a queue (Q), which is a set of nodes in the occurrence graph that need to be explored for transition firing. Q is initialized as an empty set (line 1). Graph nodes are inserted in Q as new markings are generated. $createNode()$ procedure creates a new occurrence graph node from a given marking

ALGORITHM 2: ReachabilityAnalysis

Input: CPN – Colored Petri net of selected service operations $\in S$
Input: $Init$ – Initial state
Input: $Goal$ – Goal state
Output: $T' = \{t_0, \ldots, t_q\}$ - Sequence of transitions fired to reach from the initial state $Init$ to the goal state $Goal$
1: $Q \leftarrow \emptyset$
2: createInitialMarking($Init$)
3: $m_0 \leftarrow$ getCurrentMarking()
4: createNode(m_0)
5: parent(m_0) $\leftarrow$ NULL
6: enqueue(m_0, Q)
7: **while** $m_1 \leftarrow$ dequeue(Q) **do**
8: setCurrentMarking(m_1)
9: **for all** (transition t) **do**
10: **if** (isEnabled(t)) **then**
11: fireTransition(t)
12: $m_2 \leftarrow$ getCurrentMarking()
13: setCurrentMarking(m_1)
14: **if** (isNewMarking(m_2)) **then**
15: createNode(m_2)
16: enqueue(m_2, Q)
17: **if** (isEqualMarking($m_2, Goal$)) **then**
18: **return** T' /* sequence of transitions fired in the current path from root to m_2 */
19: **return** $\emptyset$

string. Lines 2-6 compute the marking string from the initial state, creates the root node and inserts in Q. The algorithm then proceeds to perform reachability analysis by extracting markings from Q as follows: An unprocessed node (m_1) is dequeued from Q, and the current marking string is extracted from m_1 (lines 7-8). Given the current marking string, $isEnabled()$ procedure checks to see which of the available transitions can be fired (lines 9-10). If a transition is enabled (all required tokens are present in

input places, and guard evaluates true), it is fired, and a new marking string m_2 is computed (lines 12-13). The procedure $isNewMarking()$ traverses the current path to see if the fired transition has updated the system state (line 14). If so, a new node is created from m_2 and inserted in Q (lines 14-16). $isEqualMarking()$ procedure compares the new marking string m_2 with $Goal$. If $isEqualMarking()$ returns true, then the graph is traversed from the root to the current node to return the sequence of transitions fired along the path, $T' = \{t_0, \ldots t_q\}$. When the goal state is reached, the algorithm returns T'. If the goal state is unreachable, the algorithm returns an empty sequence.

4.4 Executable process code generation

Given the response process workflow synthesized through the CPN reachability analysis-based approach, the next step is to generate an executable response process. The executable process code generation component first identifies any control-flow dependencies between the different process fragments and then generates a workflow that includes parallel structures for all activities that can be executed concurrently. Based on the mapping information of each process activity to appropriate operational system/service API, the code generation component then generates executable process code (e.g., in BPEL language) of the developed workflow for deployment and instantiation on a process execution engine.

Note that multiple response organizations may have their independent deployments of the same system. For example, a fire department and a law enforcement agency both using FEMA IRIS system [3] for resource management. To reduce the state-space complexity in the CPN reachability analysis for process composition, we do not create multiple Petrinet structures for same systems of different agencies. Once we identify the service composition model for a given system, we replicate that fragment in our process workflow for multiple agencies using that system.

4.5 Deployment, instantiation and execution

In this step, the executable BPEL process generated in the previous step is first deployed on a process execution engine then instantiated and brought to execution in the runtime environment. We keep track of the process state in a collection of variables. These include, (i) User-specific variables, which correspond to user inputs and decisions taken by the user; (ii) Service-specific variables which includes all the elements in the service request and response messages; (iii) Emergency Response process-specific state information such as, response activity status (e.g., assigned or completed), response provider agencies and their assignments, resource status (e.g., requested, committed, or dispatched), incident status, and events, etc. Changes in the state act as triggers for subsequent changes in the response process.

4.6 Dynamic process re-composition and evolution

Depending on the execution results of the process (e.g., current resource availability status) as well as the dynamic changes in the environmental context due to incident evolution (e.g., a chemical leakage incident evolves into a fire incident) and user decisions, workflow of the instantiated response process may need to be extended to include additional activities. As the incident evolves, the proposed system extends and recomposes the instantiated response process by performing ontology-based reasoning to:

- search for additional organizations that can provide the resources which could not be committed by organizations contacted in the previous iteration. For example, if an air ambulance is required and is not available with the local and county-level agencies, then state level agencies may be contacted, and
- discover response activities and associated default actions, required resources, and response agencies for any new event.

The process composition modeling and code generation components in Fig. 2 are invoked again for executable code generation and redeployment of the extended response process. As shown in Fig. 2, response process extension and recomposition is performed in an iterative manner. In each iteration, the current process instance is executed to completion, and based on its output and current environmental context, the process is recomposed by adding new activities and executed again. This process extension and recomposition is continued until the incident is resolved.

We utilize a rule-based adaptive process composition approach to deal with any conflicts between the already executed part of the old process and the activities in the new composition. We assume that if there are any modifications made to the existing response process, then appropriate service APIs that allow us to perform compensatory actions (i.e., actions that would result in a partial or complete rollback and/or appropriate replacement) are available. Moreover, these compensatory actions are predefined in the rule base with respect to different events (e.g., service failure, change in the alert severity level). This is consistent with the Emergency Data Exchange Language (EDXL) messaging standards that facilitate emergency information sharing between the government entities and non-governmental organizations that provide emergency response and management services. For example in the EDXL standard for resource management (EDXL-RM) [27], if a request for a particular resource has already been placed but as the incident evolves, we realize the requested resource is not needed anymore and a different resource is required, the earlier request can be canceled by sending a request recall message which is made available through a service API. In addition, the request for the new resource need to be made with a service request to the appropriate agency.

5 Experimental Evaluation

In this section, we describe the datasets used and the experiments performed to investigate the practical usability of the proposed system. We validate the effectiveness of the proposed approach using an example scenario derived from FEMA Hazardous Materials Tabletop Exercises Manual. Specifically, we consider the scenario of hazardous chemical leakage in a train derailment incident. Response activities need to be performed based on the scenario specification.

TABLE 2: Ontology Rules Inference Execution Time

Total No. of Response Organizations	20	40	80	160	320	640	1280
Avg. No. of Resources per Organization	5	5	5	5	5	5	5
No. of Identified Organizations	9	10	33	44	44	174	464
Rules Inference Time (sec.)	6.511	8.601	8.657	8.692	15.672	36.736	157.13
Total Execution Time (sec.)	20.381	24.147	24.613	24.21	35.398	62.981	225.903

Note that the abstract response activities need to be realized through concrete services provided by the operational systems of the different response organizations. For this, we need to have the specific Web service APIs that can be used to implement different tasks such as: i) requesting resources (both material and personnel); ii) notifying commitment and deployment of resources; iii) monitoring hospital bed availability; iv) notifying hospitals of incoming patients and injury information. However, since there are no publicly accessible APIs of operational systems used for disaster management, we simulate some of the necessary APIs and make use of the available systems and services related to emergency response planning, as described in detail later.

The objective of the experimental evaluation is to show how an effective response can be composed by dynamically integrating with the systems of response organizations that may not have pre-established collaboration. Towards this, we need to measure parameters such as the amount of time it takes to identify the response organizations and to integrate with their systems, as well as any limiting factors. In specific, we evaluated the following two sub-systems:

1) Ontology-based reasoning
2) Process composition and executable code generation

Below we discuss the experiments and the datasets in each dimension. Note that our experimental evaluation does not take into account schema changes or service failures since that is not within the scope of this work.

5.1 Ontology-based Reasoning

We measure the time it takes the reasoning engine to create the new incident, identify relevant default actions, required resource types and potential response organizations, and discovering the APIs of operational systems of the identified organizations.

Dataset Description: For the experimental evaluation, we created an incident ontology in Ontology Web Language (OWL) by building on the structure of the emergency management ontology presented in [2]. We considered the train accident scenario presented in Section 1 with varying ontology domain sizes for measuring the rules inference time. We employed Apache's Jena inference subsystem [28] for reasoning and inference with our emergency management ontology.

We considered the following parameters for experimental evaluation:

- Total number of response organizations – The total number of agencies and organizations in the ontology that can potentially provide resources/services to respond to an incident on a local level, county level, state level, and federal level. We varied this number for the experiments.
- Average number of resources per organization – the resources that individual response organizations can offer.
- Number of resource types – total resource types encoded in the ontology.
- Number of identified organizations – response organizations identified as potential responders through ontology rules inference.

A total of 95 resource types and 5 resource instances per agency were registered in the ontology.

Table 2 provides the dataset statistics and computation time results for the ontology rules inference experiments. The total execution time from incident creation to ontology update includes the following: (i) reading the ontology file, (ii) creating a new incident instance, (iii) determining the default actions, and (iv) updating the ontology and saving to the file.

The 'Rules inference time', is the ontology rules inference time without considering file reading and saving time.

As depicted in the results in Table 2, the ontology rules inference time is directly correlated with the number of response organizations and resources found in the ontology. The rule inference time increases as it needs to evaluate more organizations, their jurisdictions, mutual aid agreement rules, and available resources. However, the increase is linear, and even with over 1000 organizations, requires no more than 2 minutes for inference and 4 minutes for total execution. Therefore, this is quite usable in practice.

5.2 Process Composition and Executable Code Generation

The objective is to measure the time it takes to generate the executable code of the response process from the default actions and the APIs of response organizations identified through ontology-based reasoning. This computation includes the time to run the Petri net based reachability analysis for service composition and the time to generate executable process code.

Dataset Description: Our experiment involved five response organizations' systems. We selected these organizations based on the activities in the scenario described in Section 1. Specifically, we have considered the following types of systems:

- Incident Resource Inventory System (IRIS) provided by the Federal Emergency Management Agency (FEMA) [3]. Various government agencies, jurisdictions, and communities use IRIS to inventory resources into their databases and to share information with

TABLE 3: Available Systems and Services related to Emergency Response Planning

Activity	Available Systems/Services
Resource Management	NIMS Incident Resource Inventory System (IRIS) ‡
Weather Information	Weather Web Service (Cdyne)†
Hazard Modeling	ALOHA air hazard modeling program‡
Chemicals Information	CAMEO Chemicals †
Schools Information	IES NCES Public Schools Information System †
e-Procurement	Coupa eProcurement System§, Magento§
Cargo Trains Tracking	AskRail†

§ Open source systems, † Publicly available Web services, ‡ Publicly available commercial systems

TABLE 4: Response Process Composition Time Results

Systems	R_{count}	C_{len}	T_{comp}	T_{code}
IRIS [3]	16	5	2 sec	5 sec
Webfleet [29]	18	8	2 sec	
IES Schools [30]	20	6	22 sec	
Coupa [31]	23	10	329.33 sec	
Magento [32]	26	13	53 sec	

other agencies for incident response and mutual aid purposes.

- Electronic Procurement System for requisition of relief goods from private organizations such as medicines, water, and food supplies.
- Public Schools Information System [30] for querying schools in affected areas to estimate the number of school children and staff for evacuation purposes.
- Fleet management system for transportation-related resource ordering.
- Weather and environment information services to get contextual information.
- Cargo train tracking services for rail incident response organizations.
- Hazardous chemicals information services to get response and management recommendations and hazards, such as toxic fumes.
- Plume modeling system to get threat zone estimates for various types of hazards.

The specific systems that we considered for performing our experiments are listed in Table 3.

An important point to note here is that different government agencies and organizations which provide similar services but have different jurisdictions such as city, county, or state, often use similar operational systems (e.g., FEMA's IRIS system [3] to inventory their resources and share information with other organizations). Similarly, several private organizations often use the same e-procurement system for processing their customer orders. Even if these organizations are using different systems, there is work done on API mapping, e.g., [24] that can be utilized to resolve any heterogeneity in the APIs of different organizations beforehand, as discussed in Section 4.3.

Table 4 provides the dataset statistics and computation time results for the process composition experiments. R_{count} denotes the total number of Web service operations modeled as CPN transitions in the given dataset. C_{len} denotes the number of service operations in a composition determined through the CPN reachability analysis algorithm (Algorithm 2) for a given default action such as the one shown in Fig. 6(b). T_{comp} denotes the execution time of reachability analysis based service composition averaged over three runs. T_{code} denotes the time to generate the process workflow and executable code.

Since our approach executes reachability analysis on all default actions in parallel, therefore the overall execution time for process composition is the maximum of all the compositions. Thus, the total time for the generation of the response process is 329.33 sec + 5 sec = 334.33 sec.

We can see that the reachability analysis time for service composition is much higher in the case of Coupa system as compared to IRIS and Webfleet. The main reason behind this is that the preconditions of service operations are very well defined in the case of IRIS and Webfleet. At any given state only a single operation can be applied. Whereas, in the case of Coupa, the preconditions of 3 to 4 service operations are simultaneously satisfied and because of this the number of branches in state reachability analysis tree in Coupa are way more than IRIS and others. Hence, it takes more time. An example is shown in Table 5. We can see that upon the availability of *requisition_id* and *requisition_status* in a given state (e.g., after the execution of *requistion_create*), the preconditions of three operations are satisfied simultaneously, resulting in the creation of four parallel branches in reachability tree in a single state thus incurring increased computational time. Moreover, once the execution sequence of the Web services is determined for an default action associated with a particular organization, the determined execution sequence is added to the knowledge base for future reference to avoid the need for recomputing the reachability tree for generating response processes in future incidents.

TABLE 5: Coupa Requisitions API [31].

Operation Name: *requsition_create* Precondition: {⟨requester≠"*null*"⟩, ⟨req_lines≠"*null*"⟩} Postcondition: {⟨requisition_id≠"*null*"⟩}, ⟨req_status≠"*null*"⟩
Operation Name: *submit_for_approval* Precondition: {⟨requisition_id≠"*null*"⟩}
Operation Name: *requisition_show* Precondition: {⟨requisition_id≠"*null*"⟩}
Operation Name: *requisition_getRequester* Precondition: {⟨requisition_id≠"*null*"⟩}

A typical resource requisition system involves a composition of 3 to 8 services as depicted in the case of the IRIS system. In an earlier work [24], we reviewed eCommerce and procurement business processes in several open source ERP systems. We observed that a typical procurement order process involves 3 to 14 service calls mainly for resource querying, request processing, and order/dispatch confirmation.

Combining the ontology-based reasoning and reachability analysis/code generation results, we note that the generation and deployment of an executable response process may take between 5 to 10 minutes depending on the number of response organizations involved. This time overhead for executable response process composition seems reasonable considering that the collaborating response organizations may not be known a priori and may not have pre-established system-level interoperability.

6 Implementation and Deployment

In this section, we provide a brief overview of the prototype implementation that has been developed for the proposed framework to illustrate its functionality.

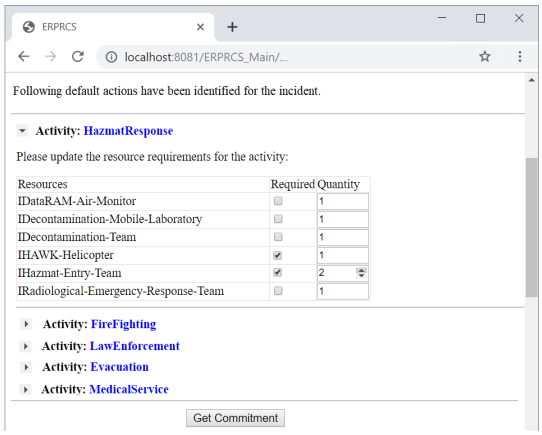


Fig. 7: Recommended response actions and resources for the freight train derailment incident

We have developed an emergency management ontology in Ontology Web Language (OWL) for the prototype system by building on the ontology presented in [2]. For reasoning and inference with the ontology, we used Apache's Jena inference subsystem [28]. The Web-based application has been developed in Java (J2EE) with Apache Tomcat Web server. Apache ODE has been used for the deployment and execution of BPEL based response processes.

The user (Incident commander) is provided an interface that facilitates the creation of a new incident in the ontology. The initial input to the system requires basic incident information as given in the incident report (incident type, date, and location, etc.). Based on this information, the system recommends default response actions along with the types of resources required, as shown in Fig. 7. The user can then modify the recommended actions and resources as required. Based upon the user's decision and the information available in the ontology, the system identifies potential response organizations and the APIs of their operational systems/ services for requesting the required resources. Upon the user's confirmation, a complete response process is generated and deployed to enable interaction between the incident command system and response agencies/organizations for: (i) checking availability status of needed resources; (ii) making resource requests; (iii) committing resources against requests; and (iv) tracking the status of committed resources.

Fig. 8 shows a partial view of the generated response process with fragments for acquiring information about the chemical hazards and assessment of any threat areas that may require rescue and evacuation operations for example due to fire potential shown in red block, requesting fire engines from local fire department shown in blue block, evacuation vehicles shown in green block from a local private company. The right side of the figure shows the execution results of the response process fragment for the assessment of hazards and threat zones. On the left, resource commitment status is presented to the user. Thus, deploying the developed system is quite seamless since the proposed framework is designed to provide additional capabilities in terms of recommending the relevant response activities as well as interfacing with the relevant operational systems of the various response organizations.

7 Conclusion and Future Work

In this article, we present an integrated approach for the on-the-fly composition of emergency response processes by enabling information sharing and interoperability among the information systems of relevant response organizations. Our proposed approach does not require any pre-established collaboration among the response organizations and employs ontology-based reasoning to identify the required response activities, needed resources, and response organizations who can provide such resources. It then uses the Web service APIs of the operational systems of these organizations to generate an executable response process to enable interaction between the incident command system and response organizations for resource management operations. We have also experimentally validated the effectiveness of the proposed approach using an example scenario derived from FEMA Hazardous Materials Tabletop Exercises Manual. The experimental evaluation separately considers the time taken for the ontology based reasoning and for the process composition and executable code generation. It is clear that the time taken to generate and deploy an executable response process is reasonable considering that the collaborating response organizations may not be known *a priori* and may not have pre-established system-level interoperability. It is worth noting that the experimental evaluation does not explicitly test the proposed approach for correctness since the evaluation is driven by the scenario and we assume that all of the necessary operational system APIs are available. Overall, the conclusions are that as long as the ontology is sufficiently comprehensive, it is indeed feasible to dynamically compose a response process based on the evolving situation. While the quality of the results is clearly dependent on the quality of the underlying ontology, evaluating this requires extensive analysis by domain experts.

In the future, we plan to conduct such qualitative evaluations by working with the emergency management community. We also plan to extend the capabilities of the proposed framework to support response process composition by learning from the knowledge of response processes instantiated for past incidents. Essentially, this involves updating the default actions based on the actual processes instantiated earlier. This may considerably reduce the response process composition time. In addition, we plan

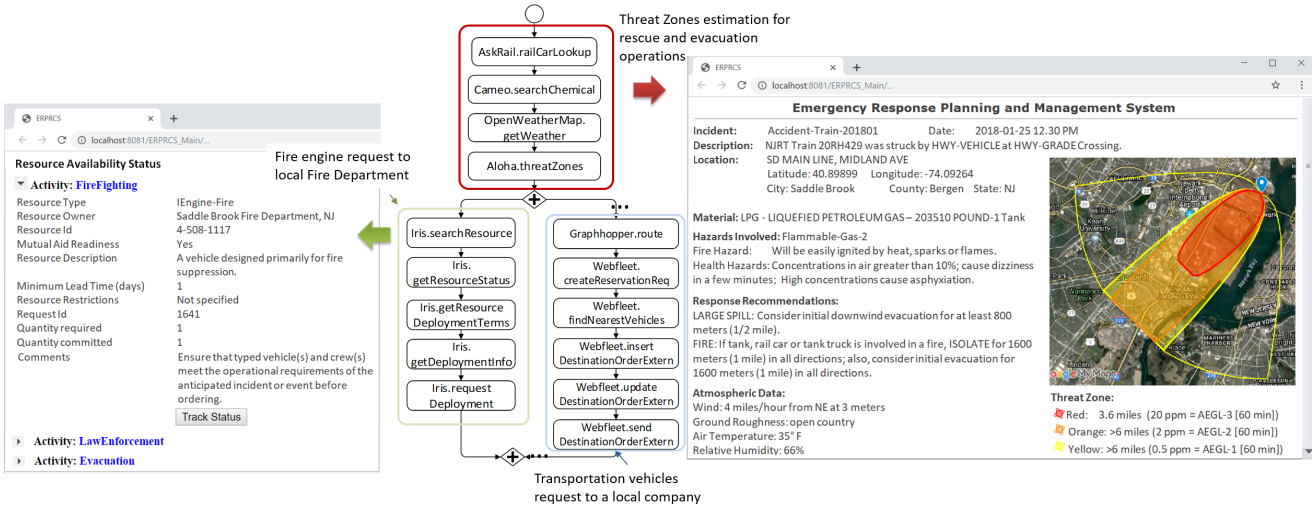


Fig. 8: Partial view of the generated response process (in the middle) and execution results (on left and right)

to evaluate the effectiveness of our proposed approach in application domains other than emergency management.

Acknowledgments

Research reported in this publication was supported by the Pakistan Higher Education Commission (HEC) NRPU Grant *P*#2305, the National Science Foundation under awards CNS-1624503 and CNS-1747728, and the National Institutes of Health under award R35GM134927. The content is solely the responsibility of the authors and does not necessarily represent the official views of the agencies funding the research.

References

[1] J. W. Morentz, "Unified incident command and decision support (UICDS): a Department of Homeland Security initiative in information sharing," in *Technologies for Homeland Security, 2008 IEEE Conference on*. IEEE, 2008, pp. 321–326.

[2] B. Shafiq, S. Ae Chun, V. Atluri, J. Vaidya, and G. Nabi, "Resource sharing using UICDS framework for incident management," *Transforming Government: People, Process and Policy*, vol. 6, no. 1, pp. 41–61, 2012.

[3] FEMA, "Incident Resource Inventory System," https://nimstools.preptoolkit.org/, 2016, accessed: 2018-04-04.

[4] S. Döweling, F. Probst, T. Ziegert, and K. Manske, "SoKNOS: An Interactive Visual Emergency Management Framework," in *GeoSpatial Visual Analytics*. Springer, 2009, pp. 251–262.

[5] N. Adam, J. Eledath, S. Mehrotra, and N. Venkatasubramanian, "Social media alert and response to threats to citizens (SMART-C)," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*. IEEE, 2012, pp. 181–189.

[6] FEMA, "FEMA Hazardous Materials Tabletop Exercises Manual," *FEMA Emergency Management Institute (EMI) Virtual Table Top Exercise (VTTX)*, July 2006.

[7] J. Yu, Q. Z. Sheng, J. K. Swee, J. Han, C. Liu, and T. H. Noor, "Model-driven development of adaptive web service processes with aspects and rules," *Journal of Computer and System Sciences*, vol. 81, no. 3, pp. 533–552, 2015.

[8] W. Song and H.-A. Jacobsen, "Static and dynamic process change," *IEEE Transactions on Services Computing*, vol. 11, no. 1, pp. 215–231, 2016.

[9] L. Sabatucci and M. Cossentino, "Supporting dynamic workflows with automatic extraction of goals from BPMN," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 14, no. 2, pp. 1–38, 2019.

[10] C. Di Ciccio, A. Marrella, and A. Russo, "Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches," *Journal on Data Semantics*, vol. 4, no. 1, pp. 29–57, 2015.

[11] N. Assy, N. N. Chan, and W. Gaaloul, "An automated approach for assisting the design of configurable process models," *IEEE Transactions on Services Computing*, vol. 8, no. 6, pp. 874–888, 2015.

[12] M. La Rosa, M. Dumas, R. Uba, and R. Dijkman, "Business process model merging: An approach to business process consolidation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 2, p. 11, 2013.

[13] J. C. Buijs, B. F. van Dongen, and W. M. van der Aalst, "Mining configurable process models from collections of event logs," in *Business Process Management*. Springer, 2013, pp. 33–48.

[14] W. M. Van Der Aalst, "Configurable services in the cloud: supporting variability while enabling cross-organizational process mining," in *On the Move to Meaningful Internet Systems: OTM 2010*. Springer, 2010, pp. 8–25.

[15] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam, "Adaptive process management with ADEPT2," in *21st International Conference on Data Engineering (ICDE'05)*. IEEE, 2005, pp. 1113–1114.

[16] A. Marrella, M. Mecella, and S. Sardina, "Intelligent process adaptation in the SmartPM system," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 2, p. 25, 2017.

[17] R. Eshuis, R. Hull, and M. Yi, "Reasoning about property preservation in adaptive case management," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 1, p. 12, 2019.

[18] A. Bucchiarone, M. Pistore, H. Raik, and R. Kazhamiakin, "Adaptation of service-based business processes by context-aware replanning," in *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–8.

[19] A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik, "A context-aware framework for dynamic composition of process fragments in the Internet of services," *Journal of Internet Services and Applications*, vol. 8, no. 1, p. 6, 2017.

[20] A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, and P. Traverso, "Incremental composition for adaptive by-design service based systems," in *Web Services (ICWS), 2016 IEEE International Conference on*. IEEE, 2016, pp. 236–243.

[21] A. L. Lemos, F. Daniel, and B. Benatallah, "Web service composition: A survey of techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 33, 2016.

[22] L. Baresi and S. Guinea, "Self-supervising BPEL processes," *Software Engineering, IEEE Transactions on*, vol. 37, no. 2, pp. 247–263, 2011.

[23] FEMA-NIMS, "National incident management system guideline for mutual aid," https://www.fema.gov/emergency-managers/nims/components, 2017 (accessed August 28, 2020).

[24] A. Afzal, B. Shafiq, S. Shamail, A. Elahraf, J. Vaidya, and N. Adam, "ASSEMBLE: Attribute, Structure and Semantics based Service Mapping Approach for Collaborative Business Process Development," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2018.
[25] K. Jensen, *Coloured Petri nets: basic concepts, analysis methods and practical use*. Springer Science & Business Media, 2013, vol. 1.
[26] A. Das Sarma, X. Dong, and A. Halevy, "Bootstrapping pay-as-you-go data integration systems," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. ACM, 2008, pp. 861–874.
[27] OASIS, "Emergency Data Exchange Language Resource Messaging (EDXL-RM) 1.0. OASIS Standard." *OASIS Emergency Management Technical Committee*, December 2009.
[28] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, 2004, pp. 74–83.
[29] T. Telematics, "WEBFLEET.connect 1.43.0 Reference," https://telematics.tomtom.com/en_gb/webfleet/partners/integration/developer-resources/, 2019 (accessed May 6, 2019).
[30] IES, NCES, "National Center for Education Statistics (NCES) Public Schools Directory," https://nces.ed.gov/ccd/schoolsearch/, 2019 (accessed May 6, 2019).
[31] Coupa, "Coupa Technical Documentation," https://success.coupa.com/Integrate/Technical_Documentation/API/Resources/Transactional, 2019 (accessed May 6, 2019).
[32] Magento, "Introduction to the Magento 1.x REST API," http://devdocs.magento.com/guides/m1x/api/rest/introduction.html, 2019 (accessed May 6, 2019).

Abeer Elahraf is a PhD student in the MSIS department at Rutgers University. Her research areas include distributed systems, Web services and big data analytics.

Ayesha Afzal is an Assistant Professor at Air University, Multan, Pakistan. Her research interests are in the areas of distributed systems, business process management and big data analytics.

Ahmed Akhtar is a PhD candidate in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. His research interests are in the areas of distributed systems and service-oriented computing.

Basit Shafiq is an Associate Professor in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. He has published over 60 research papers in international conferences and journals. His research interests are in the areas of distributed systems, security and privacy, semantic Web and Web services.

Shafay Shamail is an Associate Professor in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. He has worked both in software industry and academia. His research interests include software quality, cloud computing and e-government architectures. His publications include over 60 research papers.

Jaideep Vaidya is a Professor of computer information systems with Rutgers University and the Editor in Chief of IEEE TDSC. He has published over 140 papers in international conferences and journals. His research interests are in privacy, security, and data management. He is an ACM Distinguished Scientist.

Nabil R. Adam is a distinguished professor of Computers and Information Systems at Rutgers University. He was also the founding director of the Rutgers Institute for Data Science, Learning, and Applications and was on loan as a fellow of the US Department of Homeland Security, Science & Technology Directorate where he served as a senior program manager and a branch chief.