

The Design of XPAE - An Emergency Plan Definition Language

MARCO ANTONIO CASANOVA¹
TATIANA ALMEIDA SOUZA COELHO¹
MARCELO TÍLIO MONTEIRO DE CARVALHO²
EDUARDO THADEU LEITE CORSEUIL²
HÉRICA NÓBREGA²
FÁBIO MEIRA DIAS²
CARLOS HENRIQUE LEVY^{2,3}

¹Departamento de Informática
{casanova, tati}@inf.puc-rio.br

²Grupo de Tecnologia em Computação Gráfica / TeCGraf
{tilio, thadeu, herica, fmdias}@tecgraf.puc-rio.br, ³levy@k2sistemas.com.br

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente, 225 - Rio de Janeiro, RJ - Brasil - CEP 22.453-900

Abstract. The decisions that guided the design of XPAE, a plan definition language, are discussed. The language is one of the major building blocks of InfoPAE 4.0, an emergency plan deployment system. The language presupposes that a plan will be written as an XML document, a strategy that has well-known benefits. The overall architecture of InfoPAE 4.0 is also sketched and a brief explanation about how the language elements help achieve the desired functionality is included. The design of XPAE builds on the experience accumulated during the last two years with the implementation of a large number of emergency plans using the current version of the tool.

1. Introduction

An emergency plan comprises a structured collection of actions that must be followed to adequately respond to an anticipated set of crisis scenarios. The plan must also describe the resources required, including human and material resource, and ancillary documentation, such as maps, simulation results, lists of authorities to contact, etc. An emergency plan deployment system must therefore help monitor the execution of emergency plans and organize and access the often-vast collection of resources and documents.

This paper addresses the decisions that guided the design of XPAE, a plan definition language. The language is one of the major building blocks of InfoPAE 4.0, an emergency plan deployment system that features plans hyperlinked to georeferenced data.

The design of XPAE builds on the experience accumulated during the last two years with the implementation of a large number of emergency plans for an oil company, a pipeline operation company and gas distribution companies, using InfoPAE 3.x [4, 15]. Briefly, XPAE features special constructors that improve the legibility of plans and simplify the process of creating plans. The language presupposes that a plan will be written as an XML document [17], a strategy that has the following well-known benefits. First, the document that represents a plan offers an exchange format for the plan

and is human readable. Second, one can write a DTD (*Document Type Definition*) or an XML schema that captures a significant part of the syntax of XPAE. Lastly, the basic syntax of an XML document that represents a plan can be validated against the DTD using a variety of tools, which simplifies the implementation of XPAE editors.

XPAE shares some of the concepts of XRL (*eXchangeable Routing Language*) [1], a language that supports the exchange of workflow specifications across companies. The InfoPAE system draws some ideas from WIDE [5, 7], a project that facilitates the design of workflow specifications, separating the information about the organization from the information about resources and the process itself. The Workflow Management Coalition [8], in turn, is a non-profit organization that produced a reference architecture for the development of workflow systems.

FRIEND [3], INCA [9] and MokSAF [10] are examples of emergency management systems. In particular, the MokSAF system supports route planning by combining AI techniques with GIS. Other examples of multi-agent systems with internal planning components are RETSINA [13] and HIPaP [14]. A survey of cooperative multi-agent systems appears in [11] and [6] provides an interesting application of plan generation in the context of databases.

This paper is organized as follows. Section 2 summarizes the major concepts of XPAE, based on the lessons learned during the design, implementation and industrial use of InfoPAE 3.x. Section 3 discusses the central constructs of XPAE. Section 4 sketches the overall architecture of InfoPAE 4.0 and discusses how the language constructs help achieve the desired functionality. Section 5 presents the conclusions. Finally, Appendix I presents simple examples of the most important constructs of XPAE, while Appendix II displays sample screens of the InfoPAE system.

2. XPAE Language Concepts

The experience accumulated with the use of InfoPAE 3.x indicated that emergency plans are highly structured and follow specific standards.

For example, one of the customers of InfoPAE structures a plan into well determined phases, which are: (1) emergency identification and immediate actions; (2) mobilization of the emergency teams; (3) characterization of the accidental scenario; (4) combat procedure; and (5) closing procedure

In general, an *accidental scenario* or simply, a *scenario* is characterized by well-defined attributes, usually indicating the nature of the accident, the product involved (if applicable), the site where the accident occurred and the environmental conditions under which the accident took place. An example of a scenario would be “spill of oil type II at the pier, with high tide and south wind”.

Each scenario is associated with a *combat procedure*, consisting of elementary actions, structured according to a specific combat logic. Each action, or group of actions, may in turn be associated with people or institutions to be contacted, types of resources required to perform the action, emergency teams to be mobilized and general documentation to be used.

There also is considerable similarity, if not redundancy, between combat procedures for scenarios of the same plan. In fact, they frequently differ only on the associated information.

Users also strongly suggested that it should be possible to invoke combat procedures by clicking on the location of the accident, visualized on the user interface. This can be achieved indirectly by georeferencing the scenarios and creating a *scenario information layer*. Since combat procedures are always linked to scenarios, they also become georeferenced objects that can be activated from the scenario information layer.

In view of these observations, XPAE then includes elements to:

- define subplans
- define classes of (georeferenced) scenarios and associate combat procedures to scenarios
- structure elementary actions
- hyperlink actions, and other parts of a plan, to objects and documents in a database
- limit plan redundancy

Table 1 enumerates the major elements of XPAE, some of which are discussed in more detail in Section 3. The elements (in Portuguese, in the original language, but translated here for the sake of consistency) “do”, “ask”, “group”, “test”, “repeat”, “call”, “send”, “receive”, “characterize”, “fire” and “monitor” are also called commands.

Figure 1 schematically shows how a plan would be specified in XPAE.

Table 1 – XPAE Elements

Element Group	Elements
Related to plans	“plan”, “call”, “send”, “receive”
Related to scenarios	“type_of_scenario”, “scenario”, “class_of_scenario”, “characterize”, “fire”, “monitor”
Related to parameters	“value”, “domain”, “parameter”
Related to action control and execution	“do”, “ask”, “group”, “test”, “repeat”, “classify”, “associate”
Related to the repository	“type”, “user”, “permission”, “resource”, “style”, “envelop”, “include”

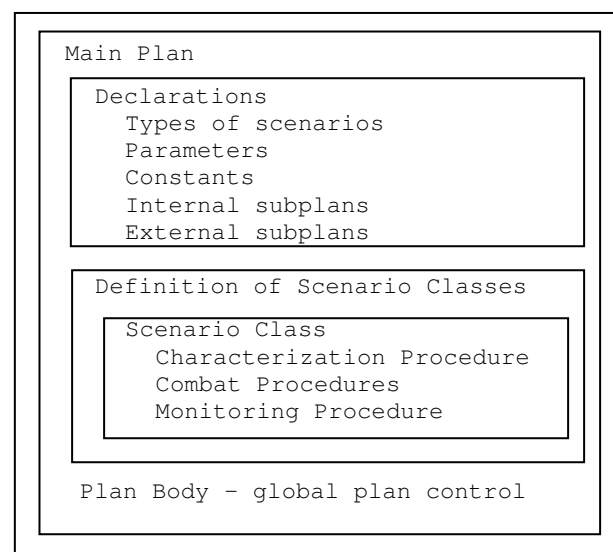


Figure 1 – Typical structure of a plan.

3. XPAE major constructs

This section discusses the central elements of XPAE, from a high level perspective. Appendix I shows a list of simple examples illustrating the elements.

3.1 Elements related to Plans

The element “plan” defines a collection of actions, structured as a workflow. To facilitate its definition, a plan may be decomposed into subplans and it may declare classes of scenarios equipped with methods (see Section 3.2).

A plan may be executed as an *independent* process and it may be executed by more than one process. If executed by a separate process, it may be *interruptible* or *cancelable*.

A subplan behaves exactly as a subroutine, callable from the plan where it was defined. Parameter passing is always by value. Without going into details, in general, scope rules are fairly rigid to avoid side-effects.

Inter-process communication uses *message queues* that intuitively model questions and answers between users of distinct processes. Sending a message does not block the sending process; but receiving a message does block the receiver until a message indeed exists in the queue. Commands “send” and “receive” implement process communication.

3.2 Elements related to scenarios

The element “type_of_scenario” corresponds to the declaration of a type of scenario that defines a list of *attributes* and, optionally, indicates a georeferenced attribute of the list. An attribute has a name and a type, which in the present implementation is just “scalar”, “georeferenced” or a type of scenario defined in the plan.

The element “scenario” describes a *scenario* of a given type T and is defined as a function C that maps each attribute A of T into a value $C(A)$, including the undefined value.

The element “class_of_scenario” declares a *class of scenarios* D , indicating:

- a type of scenario T
- a parameter V that will hold scenarios of type T
- a *monitoring procedure*
- one or more *phases*, each optionally defining:
 - a *characterization procedure* for the phase
 - one or more *combat procedures* for the phase, each of which is associated with a *firing condition*.

A scenario can be *addressed* by a combat procedure P iff the attribute values defined by the scenario make the firing condition of P true. Thus, for each phase, the class of scenarios must be such that every scenario generated by the execution of the characterization procedure of that phase may be addressed by one or more combat procedures of that phase.

A scenario for which a combat procedure P was fired is said to be *addressed* by P . A scenario is said to be *partially addressed in a phase* iff it is addressed by some combat procedure of that phase; otherwise it is said to be *unaddressed*. A scenario is said to be *fully addressed in a phase* if it is addressed by all combat procedures of that phase that can address it.

In a given execution state, the *extent* of a class is a set S of scenarios of type T . Furthermore, each scenario C in S is associated with a set of combat procedures P_C , which may be empty. All combat procedures in P_C must address C .

XPAE offers the commands “characterize”, “fire” and “monitor” to control the treatment of scenarios of a given class.

Briefly, scenario treatment may be divided into one or more phases, but it is coordinated by only one monitoring procedure.

The monitoring procedure is invoked by the command “monitor”. Before the commands in the body of the monitoring procedure are executed, a new scenario C is automatically created, with all attributes set to undefined, and assigned to the class parameter.

The characterization procedure of a phase is invoked by the command “characterize”, resulting in the definition of (some of) the attributes of the scenario that is the current value of the class parameter.

The command “fire” causes the execution of all the combat procedures of a phase that address the scenario that is the current value of the class parameter, one by one, in the order of their definition. The specific procedures, or the scenario itself, might have additional information to help guide the user.

In the current version of XPAE, these will be the only commands to treat scenarios. A future extension of the language will feature the usual class methods – *is-empty*, *insert-element*, *remove-element*, *contains-element*, *create-iterator*, *element-not-found* – and the usual iterator methods – *reset*, *next-position*, *get-element*, *no-more-elements*.

3.3 Elements related to parameters

The element “parameter” represents a parameter declaration (or program variable) to be used in a plan. In the current implementation, the parameter type can be “scalar”, “georeferenced” or one of types of scenario defined in the plan.

The element “value” defines a constant and the element “domain”, a set of constants. As a constant may be a relatively long character string, the element “value” permits associating an identifier with the constant and use this identifier to refer to the constant in other parts of the plan. A similar observation can be made for domains.

3.4 Elements related to the execution and control of actions

The commands “do”, “ask”, “group” and “test” correspond to the constructors of InfoPAE 3.x. For example, the element “do” defines an elementary action. The command “repeat”, also defined in the XRL language, represents the usual iteration construct.

The element “classify” defines classifications, whereas the element “associate” links a syntactical element of the plan with an object in the database, such as a document, report, etc. Both are applicable to a variety of syntactical elements.

3.5 Elements related to the repository

The element “repository” represents, in XML, one or more plans and objects in the database. This is the initial element of the XML document.

We comment only on two sub-elements of repository. The element “style” declares that special routines should be used to present commands or types of commands in the user interface. This element then permits (a limited) customization of the user interface.

Lastly, the element “envelop” offers the possibility of storing and retrieving syntactical elements in general preparing them for re-use. It corresponds to a primitive form of macro definition, whose expansion is governed by the command “include”.

4. Architecture of InfoPAE 4.0

We briefly discuss in this section the architecture of InfoPAE 4.0 and relate it to some of XPAE features.

InfoPAE 4.0 has the following major components:

BancoPAE	storage and retrieval of plans and data
EditaPAE	edition of plans and data publication of plans and data
OperaPAE	control of plan execution execution report generation
VistaPAE	retrieval and visualization of plans and data

This architecture is similar to that of the previous version [4], with the following major differences.

First, InfoPAE 4.0 should be viewed as a hypermedia system with two special types of object, which are plans and georeferenced data.

A typical use of the system would be:

- when detecting an accident, the user would access OperaPAE to start the appropriate plan, characterize one or more scenarios and fire combat procedures
- access VistaPAE to visualize a map of the region where an accident occurred
- display the scenarios on the map, which act as anchors to the combat procedures
- click on a scenario, select a combat procedure and invoke OperaPAE from that point to control its execution.

Therefore, VistaPAE acts as a georeferenced visual space where scenarios can be displayed and used as anchors to access combat procedures. OperaPAE should be understood as a handler to display and control plan executions, much in the way an MPEG viewer is a handler of MPEG encoded video.

This requires a tight integration of the modules since the center of incident control, to some extent, shifts from OperaPAE to VistaPAE. The XPAE language helps implementing this architecture by treating scenarios as first class objects and by clearly defining combat procedures.

5. Conclusions

This paper outlined the decisions that guided the design of XPAE. The design profited from the experience accumulated during the last two years with the implementation of a large number of emergency plans for an oil company, a pipeline operation company and gas distribution companies, using InfoPAE 3.x. It takes into account the fact that emergency plans are highly structured and follow specific standards. Moreover, they tend to be repetitive and even redundant.

With the help of simple examples, the paper also indicated how plans can be represented as XML documents.

The paper also discussed the overall architecture of InfoPAE 4.0 and briefly explained how the language elements help achieve the desired functionality.

Future implementations will explore distributed platforms, with intermittent connection. However, by considering a flexible process structure, XPAE is already prepared to support this future evolution.

Acknowledgements

We wish to thank Flávio Torres and Angelo Francisco dos Santos, from PETROBRAS, and the other members of the InfoPAE team for their many contributions to the ideas expressed in this paper.

References

- [1] W.M.P van der Aalst e A. Kumar, "XML Based Schema Definition for Support of Inter-organizational Workflow". 21st International Conference on Application and Theory of Petri Nets, Aarhus, Denmark, June 2000.
- [2] P. Bellini, R. Mattolini, and P. Nesi. "Temporal Logics for Real-Time System Specification", *ACM Computing Surveys*, Vol. 32, No. 1 (March 2000), 12--42.
- [3] B. Bruegge, K. O'Toole and D. Rothenberger, "Design Considerations for an Accident Management System". In *Proc. of the Conference on Cooperative Information Systems* (1994), 90--100.
- [4] M.T. Carvalho, J. Freire, M.A. Casanova, "The Architecture of an Emergency Plan Deployment System", III Workshop Brasileiro de GeoInformática, Instituto Militar de Engenharia, Rio de Janeiro, Brasil (4 e 5 de outubro de 2001), pag. 19-26.
- [5] S. Ceri, P. W. P. J. Grefen and G. Sanchez. "WIDE: A Distributed Architecture for Workflow Management". Seventh International Workshop on Research Issues in Data Engineering (RIDE'97), Birmingham, UK, 1997.
- [6] A. Furtado and A. Ciarlini. "Operational Characterization of Genre in Literary and Real-life Domains". In *Proc. of the ER'99 Conceptual Modelling Conference*, Paris, France (1999).
- [7] G. Sánchez Gutiérrez. "The WIDE Workflow Model and Language". Tech. Report 4111-2, May 1999.
- [8] D. Hollingsworth. "The Workflow Management Coalition Specification - The Workflow Reference Model". Tech. Report TC00-1003, Hampshire, UK, January 1995.
- [9] W. Iba, M. Gervasio. "Adapting to User Preferences in Crisis Response". In *Proc of Intelligent User Interfaces* (1999), 87--90.
- [10] T. Lenox, T. Payne, S. Hahn, M. Lewis, and K. Sycara, "MokSAF: How should we support teamwork in human-agent teams?". Tech. Report CMU-RI-TR-99-32, Robotics Institute, CMU (1999).
- [11] V. Lesser, "Cooperative Multiagent Systems: A Personal View of the State of the Art", *Knowledge and Data Engineering*, Vol. 11, No. 1 (1999), 133--142.
- [12] J. J. Moder, C. R. Phillips and E. W. Davis, Project Management with CPM, PERT and Precedence Diagramming (3rd. ed.), *Van Nostrand Reinhold*, New York (1983).
- [13] M. Paolucci, D. Kalp, A. Pannu, O. Shehory, and K. Sycara. A Planning Component for RETSINA Agents. In *Agent Theories, Architectures, and Languages* (1999), 147--161.
- [14] M. Paolucci, O. Shehory, and K. Sycara. "Interleaving planning and execution in a multiagent teamplanning environment." Tech. Report CMU-RI-TR-00-01, Robotics Institute, CMU (2000).
- [15] TecGraf, 2002, "Sistema de Informação para Apoio a Planos de Ação de Emergência. Manual de Operação e Edição", Laboratório de Computação Gráfica - TecGraf, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.
- [16] TecGraf, 2002, "InfoPAE 4.0 – Especificação da Semântica da Linguagem para Definição de Planos", Laboratório de Computação Gráfica - TecGraf, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.
- [17] W3C Recommendation. Extensible Markup Language – XML 1.0 (Second Edition), October 2000. (<http://www.w3c.org/TR/REC-xml>).

Appendix I - Examples of XPAE constructs

```
<!--Declaration of an internal plan -->
<plan name="Identification of the emergency and immediate actions"
      out="c_region"
      persistent="true"
      independent="true">
  <parameter type="georeferenced" name="c_region"/>
  <body>
    <group type="parallel"
          description="Identification of the emergency">
      <do description="Receiving the alert">
        <classify bd_id="317"/>
        <associate type="form" bd_id="458"/>
      </do>
      <do description="Stop operations at the site">
        <classify bd_id="316"/>
      </do>
      <ask description="Where did the accident occurred?"
           parameter="c_region"/>
      <do description="Send observer to the site to confirm the emergency"/>
    </group>
  </body>
</plan>
```

Example 1 – Declaration of an internal plan.

```
<type_of_scenario id="10010" name="main" georeferenced="local">
  <attribute id="100101" type="georeferenced" name="region" domain="1001"/>
  <attribute id="100102" type="scalar" name="nature" domain="1002"/>
  <attribute id="100103" type="georeferenced" name="origin" domain="1003"/>
  <attribute id="100104" type="scalar" name="type_accident" domain="1004"/>
  <attribute id="100105" type="scalar" name="type_product" domain="1005"/>
  <attribute id="100106" type="scalar" name="qty_spilled" domain="1006"/>
  <attribute id="100107" type="scalar" name="wind" domain="1007"/>
  <attribute id="100108" type="scalar" name="tide" domain="1008"/>
</type_of_scenario>
<!-- Scenario Declarations -->
<scenario id="2" type="main">
  <value attribute="region">Terminal Area</value>
  <value attribute="nature">Spill</value>
  <value attribute="origin">Pier</value>
  <value attribute="type_accident">Collision with pier</value>
</scenario>
<scenario id="3" type="main">
  <value idref="2"/>
  <value attribute="type_product">Oil type I</value>
</scenario>
```

Example 2 – Declaration of a type of scenario and constants describing scenarios.

```

<class_of_scenario type="accident" parameter="c_accident">
  <characterization>
    <ask description="Where did the accident occurred?"
      parameter="c_accident.local"/>
    <ask description=" What is type of the accident?"
      parameter="c_accident.type"/>
  </characterization>

  <combat>
    <condition>
      <eq left="c_accident.type" right="explosion">
    </condition>
    <do description="Stop operations at the site of the accident"/>
    <do description="Call the fire department"/>
  </combat>

  <combat>
    <condition>
      <and>
        <eq left="c_accident.type" right="spill">
        <eq left="c_accident.local" right="pier">
      </and>
    </condition>
    <do description="Launch contention barriers along the pier"/>
  </combat>

  <combat>
    <condition>
      <and>
        <eq left="c_accident.type" right="spill">
        <eq left="c_accident.local" right="deposit">
      </and>
    </condition>
    <do description="Stop pumping oil"/>
    <do description="Execute cleaning procedures"/>
  </combat>

  <monitoring>
    <group>
      <ask description="Is the emergency confirmed?"
        parameter="active"/>
      <test id="">
        <condition>
          <eq left="active" right="yes"/>
        </condition>
        <then>
          <characterize/>
          <fire/>
        </then>
        <else>
          <do description="Register the decision"/>
        </else>
      </test>
    </group>
  </monitoring>
</class_of_scenario>

```

Example 3 – Declaration of a class of scenarios.

Appendix II - Example of the interaction between VistaPAE and OperaPAE

The screenshot displays the VistaPAE software interface. The top menu bar includes 'Consultas', 'Ferramentas', and 'Plano'. The title bar reads 'Gerência da Baía de Guanabara - system' with coordinates and time. A left sidebar shows a layer list with checkboxes and symbols for 'Dutos', 'Instalações Ilha Redonda', 'Planta Ilha Redonda', 'Sensibilidade Costeira - linhas', 'Barreiras PP/Sul<500', and 'Recolha Dinâmica - PP_S<500'. The main map area shows a coastal region with various colored overlays and a red line. A text box 'tempo: 1 h' is visible on the map. A context menu is open over the map, listing 'Cleaning procedure' and 'Contention procedure'. Below the map, a panel shows a list of activities, including 'Parar operações envolvidas no local' and 'Área de Influência entre a Ponte do Barão e a Ilha d'água'. At the bottom, a grid of six satellite images is shown, with a yellow arrow pointing to a specific location in the bottom-right image.

1. User starts working with VistaPAE, selecting a number of layers.
2. User points to a location on the map.
3. User browses the combat procedures that can address accidents at that point.
4. User further characterizes the accident.
5. InfoPAE invokes the appropriate procedure.
6. User interacts with OperaPAE, on a separate screen, to control the execution of the plan.