BUILDING AN EMERGENCY MANAGEMENT APPLICATION AS A PROCESS DRIVEN TOOL COOPERATION

Valérie Lavigne, Marc Firmignac
Cap Gemini Innovation
86-90, rue Thiers.92513 Boulogne-Billancourt. France
Voice: 33-1-49105115
Fax: 33-1-49100615
E-mail: Valerie.Lavigne@capsogeti.fr Marc.Firmignac@capsogeti.fr

KEYWORDS: Emergency Management, Tool Integration, Workflow, Information Access

ABSTRACT

Building an Emergency Management Information System (EMIS) should be guided with at least the following requirements: providing the emergency managers (EMers) with the best services for managing the crisis situations, and especially a direct access to the information needed to act and take decisions; providing a system that is not a proprietary one but a combination of "Of The Shelf" products that work together in an efficient way; providing help and guidance to the end-user in using the system and navigating through it.
The integration of several tools together in order to build a final application should be done along four axes : data integration, control integration, presentation integration and process integration. In order to fulfill these requirements and to cover the four aspects of integration, we propose to use a Broadcast Message Server approach to define an integration framework and a Process Model approach to ease the navigation in the system.

REQUIREMENTS FOR AN EMERGENCY MANAGEMENT INFORMATION SYSTEM

An Emergency Management Information System (EMIS) that is well adapted to Emergency Managers's needs (EMers) should fulfill the following requirements:

- provide a well integrated set of services ranging from communicating with people, through evaluating a current situation and predicting the evolution of a problem, to following the activation of a contingency plan;
- easy maintenance where the replacement or the addition of functionalities is done without great modifications of the system;
- inclusion in the EMIS of existing systems: usage of a specific Geographic Information System (GIS) and its cartographic database, access to a chemical database, dedicated communication tool.
- inclusion in the EMIS of some Commercial Off The Shelf (COTS) products which are well suited to provide a subset of the EMIS functionalities : GIS, document editor, mailing tool;
- provide a support for the end-user to navigate in the EMIS set of functionalities to help him in his decision making process.

In order to take into account all the previously stated requirements in the development of an EMIS, we present in this paper the benefit of viewing the system as a tools-cooperation environment where a process driven approach provides great added value. After having described the system we have developed for a flood application in the French Alps, the technical solutions we have adopted are presented.

FLOOD MANAGEMENT IN THE FRENCH ALPS

The **MEM*brain*** project [1] facilitates the development of EMIS by providing ready-to-use services for crisis management and an integration plateform to build final applications. Within the project, CAP GEMINI INNOVATION is in charge of providing this integration plateform. We are also working on an application dealing with flood management.
In the **French Alps of Savoie** near the city of **Chambery** there exists a flood risk of the **Leysse** river. This risk is due to the proximity of mountains, and of sudden thaw of snow on the ground which, when combined with rain, creates floods.
This paper focuses on the services available for crisis management. These services are dedicated to the Chambery flood EMers. In order to make these services easier to understand, we present them in a potential flood scenario.
Within this flood scenario, support is provided for the following user's actions :

- **consultation of historical floods**: this provides cartographic views, diagrams and textual description of past floods.
- **evaluation of the risk**: the end-user indicates the risk level of a potential flood to the system (we have

classified the risk as "normal", "alarm", "alert", and "crisis"). According to this risk level the system identifies a set of actions to be performed.

- **monitoring sensors**: this provides him with real-time diagrams of the sensor data and the sensors localization on the map.
- **simulation of the flood**: this provides him with a step by step simulation of the flood evolution on cartographic views.
- **activation of a contingency plan**: the end-user can activate actions with a "check list" mechanism attached to each action in the plan.
- **annotation of the map with an incident**: this allows the end-user to follow the evolution of the situation directly annotating a map.
- **definition of snapshots of the crisis**: with a simple click on the corresponding service, the current screen image is stored in the system log.

In the presentation of the technical solutions adopted to build this system, some specific points of the system will be highlighted and described more deeply.

INTEGRATION

A classic software development life cycle is divided into several phases : a requirement phase, a design phase, a coding phase and so on. In our approach we propose to replace as much as possible the coding phase by an *integration phase* consisting of the integration of existing pieces of code like COTS products. The EMIS then becomes a well adapted "mix-and-match" of existing components.

The integration of several software components, in order to be comprehensive, must be considered along four axes :

Data integration Data integration is concerned with the data sharing (i.e. when two different components share the same data). When a product directly uses the data produced by another, there is a natural continuity and no loss of information caused by the translation of the data.

Control integration Control integration is concerned with the operation sharing, when one component is able to call an operation implemented by another component. This gives to the end-user the feeling that he is in front of one unique set of operations without knowing which product implements which operation.

Presentation integration Presentation integration is concerned with the fact that all the products which compose the application present the same look and feel.

Process integration Process integration is concerned with the sequencing of the operations may be implemented by different components. This sequencing is modeled outside the component in reference to a process which describes the end-user way of working.

Because we need to provide an environment which is comfortable and effective for the emergency manager, we have to define an architecture which addresses **all** these aspects:

- **data** : same data are shared in different components (i.e. when a document treats a specific building we want to be able to show it on the map)
- **control** : the operation of a specific component can be directly executed from a different component (i.e. when we are on the geographical view we want to be able to see the preparedness plan attached to this sensitive site)
- **presentation** : views and operations from different components have to be displayed on the same screen and so accessing them should be done in a consistent way, ergonomic rules are therefore preponderant in the environment (i.e. each component operation label is displayed with the same font and color and accessible in the same manner)
- **process** : the same component can be used for running different processes and provide many operations, so we want for the end-user a short and easy way to execute well defined processes (i.e. we want for all operations consultating past floods to be directly accessible when needed with no need to find the operation in the sub-menus of the GIS and EDITOR components)

Architecture

The architecture underlying our system addresses all these aspects of integration (data, control, presentation and process). For each we will present the baseline technology which supports it.

The architecture of the system is designed around an *integration platform* where components are *plugged in.*

Most of the functional components needed for the application already exist in the market. For example, there is in our application a GIS component which displays geo-referenced objects on a map, and an EDITOR component for document management. The architecture of the system is a collection of such component tools. A tool is : "*runtime software which manages its own user interface (displays views and provides end-user operations)*". Some tools are products available on the market and some tools are specially developed in order to provide a novel set of functionalities.

An application is defined as "*a set of tools cooperating by providing services to each other in order to achieve end-user functionalities*".

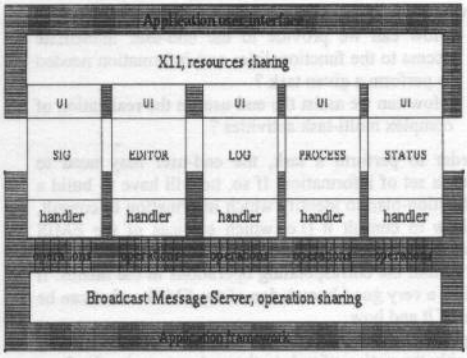

Figure 1: Architecture

In order to develop an application it is needed to build an integration framework which enables these tools to communicate between them. This integration framework defines an abstraction of each component in terms of the operations a tool-component is able to provide to another component. Then the total application is seen as a set of components, some being clients and some servers. Our integration will consist of establishing the relations between the clients and the servers. This aspect will be described in the BMS section.

This architecture allows the definition of the application through the *composition of customizable, ready-to-use bricks*. We have an easy assemblable set of components and an integration framework.

Building a new application will consist of choosing the right components according to the operations set required and then plugging the components into the integration framework.

The visible user-operations and how these operations should appear (in which view or menu of which component) may imply interaction between different components. Data exchange between components or the calling of one component from other one constitute the customization phase of the integration of the components into the framework.

As such, an open architecture is very important for our environment where the integration of a new component or the replacement of one by another has to be done easily. Managing the crisis is a complex mechanism where big volumes of information have to be taken into account but where the procedures are not fully defined nor is the information to be managed fully grasped. So an *iterative* approach should be used for building the system. And our architecture can be of great help with this approach where the integration into the environment of new components is done without a big reorganization of the total system.

Broadcast Message Server

The Broadcast Message Server (BMS) is a software developed by Cap Gemini Innovation for enabling the communication between several different tools. Originally it implements the *integration by control* which has been discussed in the Integration section (allowing a tool to call an operation implemented by another tool). We have also used the BMS for data integration. In fact we do not address data sharing but implement instead data exchange mechanism. No great volume of data being shared by different tools, tools exchange the data on explicit request instead of sharing the same database. Thus the two aspects of the integration are implemented by a single mechanism. This mechanism is supported by the BMS.

Originally the BMS is a common service for the broadcasting of messages between different tools located on a local network. There are two different types of messages, the "**request**" and the "**notification**". A tool is able to receive a message when it has defined a filter which matches the broadcasted message. A message is composed of fields identifying an operation and the tool which provides it.

For example the EDITOR requests the GIS to display on the map the most exposed flood areas. The C code for this is :

```
SendMessage("R","GIS","DISPLAY_AREA","*","FLOO
D_EXPOSURE")
```

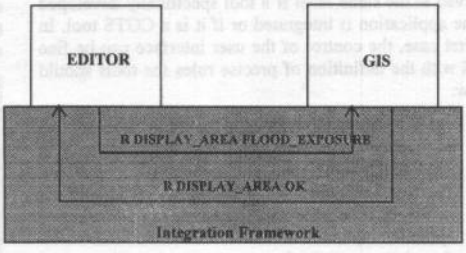

Figure 2: BMS messages

The callbacks define by the GIS server to the operation DISPLAY_AREA will be started on the reception of the message. And the GIS will notify the execution by sending a notification message with the return code in the data field.

All the "plugged" tools have to identify the operations they want to provide to others. The BMS then provides a mechanism which allows all the tools to request a query and get the result.

In order to provide an architecture which is as open as possible and to allow the interchangeability of the tools, the format of the exchanged data (like **FLOOD_EXPOSURE** in the example) is independent of the tools which are communicating them. Thus *it is*

possible to replace a tool by another one implementing the same functionalities without any effect on the rest of the environment.

The BMS defines an architecture where the tool does not appear directly as such. The only declared entities are a set of classes (like **GIS** in the example) which provide a set of operations (like **DISPLAY_AREA** in the example). When a tool wants to communicate information with another one, the only thing it has to know is the format of the message to be exchanged. The BMS delivers the message to the tool which provides the operation for its execution.

The integration of market product is done by the definition of a *handler* which presents the tools' BMS interface to the integration framework. The role of the handler is to translate the BMS operation into the internal language of the product. Depending on the openness of the product (most of the market products offer the ability to communicate through RPC, SIGNAL, SOCKET and so on) the technique for their integration will be different and the product will be integrated in more or less fine-grained manner.

USER INTERFACE INTEGRATION

Integrating several tools may be badly perceived if the resulting user interface of the application is not consistent. Indeed the end-user should not have the feeling of several tools running concurrently. The application should have a consistent look and feel. This consistency cannot be achieved at the same level if a tool specifically developed for the application is integrated or if it is a COTS tool. In the first case, the control of the user interface can be fine tuned with the definition of precise rules the tools should follow:

- for the presentation: like the font to be used in push button, the position of the "help" push button in a window ...;
- for the behavior: like a confirmation should be asked of the end-user before any destructive operation ...;
- for the customization at integration time: like changing the language, the color of the background of the windows ...

Concerning COTS tools the adaptation of their user interface is dependent on tool flexibility. Nevertheless, it is almost always possible to influence the size and position of the windows as well as some general attributes like background color. These minimal adaptations will ensure a correct level of integration since they will reduce the differences that exist between the tools.

PROCESS INTEGRATION

The functionalities dedicated to crisis management are unlikely to be very familiar to the end-user since they are not used very often. At the same time, as crisis situations are very stressing the system should be as easy to use as possible. For this reason, we have tried to find solutions to the two following points :

- How can we provide to the end-user immediate access to the functionalities and information needed to perform a given task ?
- How can we assist the end-user in the realization of complex multi-task activities ?

In order to perform a task, the end-user may need to consult a set of information. If so, he will have to build a small action-plan to identify which information to consult, and how to consult it (i.e. which services of the EMIS should he activate). Once this action plan is ready he will have to find the corresponding operations in the menus. It requires a very good knowledge of the EMIS : what can be asked of it and how.

Symplifying the system use is achieved by introducing explicitly the notion of task to the end-user under the form of a "**Work Context**". A work-context contains:

- a textual description of the task itself,
- the needed information to be consulted and the way to consult it,
- the result to be produced if any.

Let us see an example taken from our system. When the end-user receives the alarm message announcing a meteorological problem, his task will consist of answering the question : what kind of flood can be expected ? In order to do so, he will activate the task : "Consult floods" (Consultation crues). This task is presented as a work context (see "Consult historical floods task" figure).

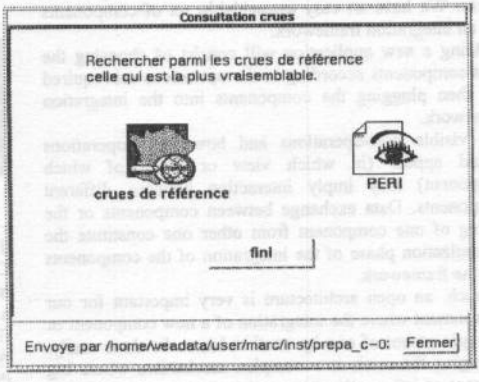

Figure 3: Consult Historical Floods Task

The purpose of the task is described and the needed information is proposed : the historical floods (*crues de reference*) in a cartographic form and the document (*PERI*) in which a textual description of them is provided.

In order to access this information, the user will simply click on the corresponding icon.
As we can see the way to access to the information is task-driven and therefore very natural for the end-user. He does not have to know which computer function to activate and how to go about it.
This provides an answer to the first requirement "*How can we provide to the end-user an immediate access to the functionalities and information needed to perform a given task ?*".
For the next point "*How can we assist the end-user in the realization of complex multi-task activities* ?", the problem is that in crisis management the EMers have many activities to managed in parallel and it can become quickly very difficult to control all these activities and to know what is going on and what is to be done next. In order to help them in their job, we have added two more elements to the EMIS:

- a list of the on-going tasks in an Agenda window;
- a process support to decide which new tasks need to be carried out.

The Agenda (see figure 4) is a graphical representation of the list of tasks to be done. A task which is in the agenda is "on", that is to say that the end-user has activated this task but it is not finished yet. When clicking on the task icon, the corresponding work context is displayed.

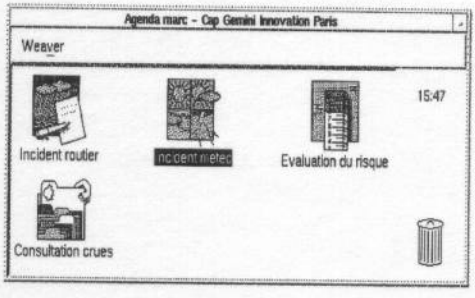

Figure 4: Agenda

When the task is finished, it is removed from the Agenda.
Having the list of tasks to be done always at hand allows the EMers to have a clear view of the situation. It is then up to them to decide which task is to be activated next. This is very important since it allows the end-user to switch from one task to another. He is not obliged to finish one task to start another. Thus, the end-user is able to delay some tasks (without abandoning them) to face the emergency.
In crisis management, there are some "reflexive activities". Such activities are activated in **known circumstances** by the end-user (after a given event has occurred for instance) or by the system itself (in case of periodic activities). The main feature of such an activity is that the sequencing of the tasks involved in it is well known.
In order to free the EMers of having a precise knowledge of these activities, the EMIS can act as a secretary reminding periodic tasks to the end-user and ensuring that all tasks of an activity are carried out.
To do this a **Process Model** approach has been used. We propose to model the reflexive activities as procedures and then to execute the procedures. This has been done using the **Process Weaver** tool which allows to model the procedures graphically through a Petri Net formalism. It also supports the definition of the work contexts previously described and the Agenda. The **Process weaver** tool supports the execution of the procedure by putting new tasks in the Agenda as described in the procedure model.
For instance, in the system a reflexive activity called **Flood preparation** has been implemented. This activity, which is activated by the end-user when a meteorological problem is announced, consists of :

- putting in the Agenda the **Consult floods** task;
- then putting in the Agenda the **Evaluate risk** task;
- then if the risk level has been set to "*alarm*" putting in the Agenda the two new tasks :
  1. "*Send an alarm message to the corresponding authority*"
  2. "*Sensors monitoring*".

The advantage of this solution is that the system is not too constraining. Once a task is in the Agenda it is the end-user's decision to decide to perform it or not.
We must be very careful when using this kind of approach for a crisis management system. Not ALL activities need to be or can be modeled as procedures.
Process integration is interesting for EMIS for two reasons :

1. in the development of an EMIS this can be done progressively : once the needed functionalities of the EMIS are implemented a "*process level*" can be added to ease the EMers's job;
2. it provides great help to the user in navigating in the EMIS.

CONCLUSION

As we have seen throughout this paper, building an emergency management application as a process driven tool cooperation is a profitable approach for at least three reasons :

- putting user's needs before technical constraints : what services are really needed ? How can he use them in the best way ? How can he access them in the easiest way ?
- adapting software use to the EMers's natural way of working : tool cooperation allows the definition of systems which are closer to the existing world

and the natural way of working (the very computer aspect of the system is not made so evident).

- reduction of development time since it gives a greater importance to the reuse of existing up to date software.

REFERENCES

[1] The **MEM*brain*** project is part of the EUREKA program and is composed of the partners : QUASAR, IFE and OCEANOR (Norway), IFAD and RISO (Danemark), VTT (Finland), CAP GEMINI INNOVATION (France).

[2] Shavit Y., 1994. *Integration of Emergency Information System : Towards a Common Reference Model.* The International Emergency Management and Engineering Conference, April 18-21, Hollywood Florida Beach.

[3] Kvalem J., 1994, *Integrated User Interface for MEM decision Support*, The International Emergency Management and Engineering Conference, April 18-21, Hollywood Florida Beach.

[4] **ECMA**, April 1993, *Reference Model for Frameworks of Software Engineering Environments.*

[5] *Process Weaver : User's Manual*, Version PW 2.0