FORMALISATION OF A FUNCTIONAL RISK MANAGEMENT SYSTEM

Víctor M. Gulías, Carlos Abalde, Laura M. Castro and Carlos Varela
MADS Group. Computer Science Department
University of A Coruña (Spain)

Keywords: Risk management, functional programming, business process modelling.

Abstract: This article shows a first approximation to the formalisation of a risk management information system. It is based on our experience in the development of a large, scalable and reliable client/server risk management information system.

1 INTRODUCTION

In this paper, we present our first attempt to formalise the model of a large, scalable and reliable client/server risk management application called ARMISTICE (*Advanced Risk Management Information System: Tracking Insurances, Claims and Exposures* (Gulías et al., 2005) http://mads.lfcia.org/armistice).

The main novelty of this system, in the field of risk management information systems (RMISs), is the new approach used to manage the different kind of heterogeneous business objects. ARMISTICE provides an abstract framework that can be easily adapted to any organisation. It represents a step forward compared with other popular RMIS products like (STARS, 2005) or (AON, 2005).

The paper is structured as follows. First, a few basic knowledge about the whole system architecture is presented. Then, section 3 provides the non-technical formalisation of the system internals. Finally, section 4 shows some conclusions.

2 SYSTEM ARCHITECTURE

ARMISTICE has a client/server architecture, structured in layers using two well-known architectural patterns: Layers and Model-View-Controller (Marinescu, 2002). Figure 1 provides an overview. The user side is a lightweight Swing/Java client which performs remote procedure calls to the server (XML-RPC) and has no associated logic. The server side, completely developed in the distributed functional declarative language Erlang (Wadler, 1998; Armstrong et al., 1996), supports the model and all the business logic. In (Cabrero et al., 2003), some additional technical information can be found.

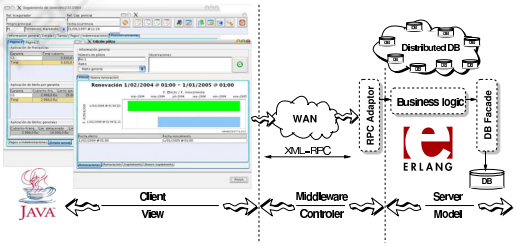

Figure 1: ARMISTICE three tier architecture.

3 INSURANCE POLICY MODEL

Our system is split into three large subsystems: *risks*, *policies* and *claims*.

3.1 Risks Subsystem

3.1.1 Risks Situations and Their Attributes

The key concept in this subsystem is the *risk situation* (RS). A RS models the state of any element that can be included on the coverage of a policy (e.g. a shop or a vehicle). Every RS is an instance of a *risks group*

M. Gulías V., Abalde C., M. Castro L. and Varela C. (2006).
FORMALISATION OF A FUNCTIONAL RISK MANAGEMENT SYSTEM.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 516-519
DOI: 10.5220/0002496205160519

(RG). A RG is a template, that is, a meta-description of all the important attributes (e.g. address, vehicle registration number, warehouse area or cubic capacity) which describe the state of any element that can be insured (e.g. commercial locals or vehicles).

We denote the set of attributes as $\mathcal{A}$, each of them defined as $Attribute = (name, type)$ where

$$name \in String$$
$$type \in Types \quad = \quad \{String, Boolean, \dots\}$$

Over these elements, the set of RGs $\mathcal{RG}$ is defined. Each member $\mathcal{RG}_i \in \mathcal{RG}$ is a set like $\mathcal{RG}_i = \{a_0, a_1, \dots a_{n-1}\}\ \forall x \in [0, n-1]\ a_x \in \mathcal{A}$.

Therefore a RG is a set of attributes which provides a meta-description of any property that can be important from the insurance's point of view.

Finally, using the $\mathcal{RG}$ set, the set of RSs (that is, insurable elements) $\mathcal{RS}$ is defined. A RS $\mathcal{RS}_j$ is an instance of a RG $\mathcal{RG}_i$ ($\mathcal{RS}_j : \mathcal{RG}_i \longrightarrow \cup\ Types$). Every attribute in the RG has a concrete value assigned in the RS.

$$\mathcal{RS}_j = \{p_0, p_1, \dots p_{n-1}\}\ \forall x \in [0, n-1]\ p_x \equiv (a, v) \\ \wedge\ a \in \mathcal{RG}_i \wedge\ v \in Type(a)$$

Then, a RS represents the state of any element that can be insured. For short, we will use $\mathcal{RS}_j.a$ to denote the value of attribute a of the RS $\mathcal{RS}_j$.

3.1.2 Versions and Revisions of RSs

Using the previous abstraction, it is possible to model the state of every insurable element (i.e. RSs). However, business requirements demand a temporal representation of the RSs. Based on Fowler's patterns (Fowler, 2005), a temporal tracking of the $\mathcal{RS}$ set is performed. The evolution of a RS is modelled since its creation as a set of *versions* and *revisions* (two-dimensional temporal modelling). A *version* represents a new state of a RS as a result of a business event. However, a *revision* represents a new state of a RS as a result of a mistake or any other business external event. This behaviour is provided modifying the original definition of a RS as follows:

$$\mathcal{RS}_j = \{\mathcal{RS}_j^{v_0}, \dots \mathcal{RS}_j^{v_{\alpha-1}}\} \\ \forall x \in [0, \alpha-1]\ v_x \in Timestamp \wedge \\ v_x < v_{x+1} \wedge \\ \mathcal{RS}_j^{v_x} = \{\mathcal{RS}_j^{v_x, r_0}, \dots \mathcal{RS}_j^{v_x, r_{\beta-1}}\} \wedge \\ \forall y \in [0, \beta-1]\ r_y \in Timestamp \wedge \\ r_0 = v_x \leq r_y < r_{y+1} \wedge \\ \mathcal{RS}_j^{v_x, r_y} = \{p_0, p_1, \dots p_{n-1}\} \wedge \\ \forall z \in [0, n-1]\ p_z \equiv (a, v) \wedge \\ a \in \mathcal{RG}_i \wedge \\ v \in Type(a)$$

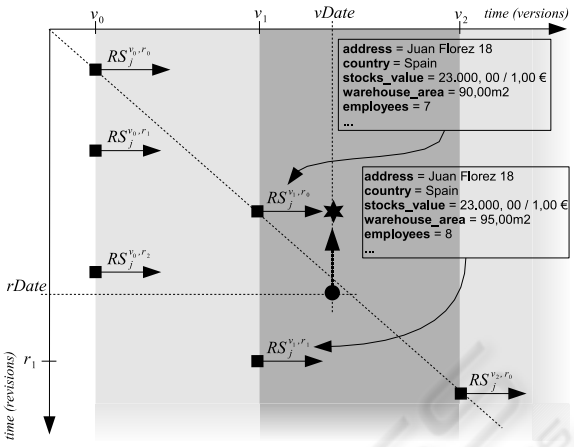

Figure 2: Versions and revisions of a RS.

So the elements belonging to the $\mathcal{RS}$ set can be defined as a collection of items representing different states from different RSs:

$$\mathcal{RS} \equiv \{\mathcal{RS}_0^{v_0, r_0}, \dots, \mathcal{RS}_j^{v_x, r_y}, \dots\} \\ \forall j \in [0, |RS|-1] \wedge \\ x \in [0, \alpha_j - 1] \wedge \\ y \in [0, \beta_{j,x} - 1]$$

where $|RS|$ are the actual different modelled RSs.

Therefore, the expression $\mathcal{RS}_j.a$ must be changed by $\mathcal{RS}_j.a[vDate][rDate]$. With this new model questions like "At the moment in time $rDate \in Timestamp$, what was the value we *thought* attribute a (of RS $\mathcal{RS}_j$) had at date $vDate \in Timestamp$, and which one we do know it is now?" can be answered. Figure 2 shows a schematic view of the evolution of the versions and revisions of a RS $\mathcal{RS}_j$ through time.

3.1.3 Hazards that Affect RSs

The risks subsystem is not only a tool for modelling the state on any insurable element. There is an additional and very important concept: the *hazards*. A *hazard* can act over a RS causing a damage or accident. The system has to manage a set of hazards $\mathcal{H}$ (e.g. fire, explosion or terrorism). Each business model must state a standard list of hazards adapting it to its domain, to ease policy modelling.

Thus, ARMISTICE allows the user to handle the sets $\mathcal{A}$, $\mathcal{GR}$, $\mathcal{RS}$ and $\mathcal{H}$. By doing so, our application can be adapted to a broad variety of business models: the first step would be the definition of attributes and RGs. Then, not only the RSs would be created over these abstractions, but any relevant change in their states, too. These modelling and tracking activities are the fundamental working basis for the rest of the system.

3.2 Policies Subsystem

Policies are the most complex element in the system. They group and link every business object. ARMISTICE works over a set of policies $\mathcal{P}$, where each policy $\mathcal{P}_i$ is modelled as a set of renewals $\mathcal{P}_i^{r_j}$. A *renewal* represents a new policy created to provide coverage to a new set of RSs in a new temporal interval. At the same time, a renewal is composed by a set of supplements $\mathcal{P}_i^{r_j,s_k}$. A *supplement* represents a *revision* of the policy to change its coverage, to change its contractual clauses, to change its relevant dates, etc. A supplement represents the minimal element that can be used to give coverage to a claim. Here, a temporal modelling is applied, like in section 3.1.2, but with three dimensions or temporal axis: each supplement $\mathcal{P}_i^{r_j,s_k}$ is parametrised by starting and ending validity dates, an activation date and a timestamp.

$$\mathcal{P}_i \equiv (Number, \{\mathcal{P}_i^{r_0}, \dots \mathcal{P}_i^{r_{\alpha-1}}\})$$
$$\mathcal{P}_i^{r_j} \equiv (Receipts, \{\mathcal{P}_i^{r_j,s_0}, \dots \mathcal{P}_i^{r_j,s_{\beta-1}}\})$$
$$\mathcal{P}_i^{r_j,s_k} \equiv (Start, End, Activation, Timestamp, Coverage, Conditional) \wedge Coverage \subseteq \mathcal{RS}$$

where $Start, End, Activation, Timestamp \in Timestamp$ are the dates used to determine the location of the validity period of the policy in the axis of the three dimensions temporal modelling. As far as the $Coverage$ is concerned, it represents the collection of RSs states building the insurance of the policy. Of course, this means that every $\mathcal{RS}_j^{r_x,s_y}$ inside the $Coverage$ set is different, $Coverage \subseteq \mathcal{RS} \quad \forall \mathcal{RS}_{j_1}^{v_{x_1},r_{y_1}}, \mathcal{RS}_{j_2}^{v_{x_2},r_{y_2}} / j_1 \neq j_2$.

Each policy renewal $\mathcal{P}_i^{r_j}$ is related to an element we have called $Receipts$. This component is in fact a receipt model ($ReceiptModel$), and a set of receipts that appear as a result of some hazard affecting one or more of our RSs ($\{R_1, \dots R_n\}$), $Receipts \equiv (ReceiptModel, \{R_1, \dots R_n\})$.

When an accident happens to a RS (e.g. a shop), and we have a particular policy $P_i^{r_j,s_k}$ to cover all the arrangements and repairs to be done, all those payments can be broken down into several items. That is what the $ReceiptModel$ is intended for: using formulas-based models (see 3.2.2), it is possible to express the calculation for each item, and so it is possible to foresee the amount of all the receipts, even before they arrive ($R_1 \dots R_n$).

3.2.1 Conditional Clauses

A *conditional* is the key object to implement the ARMISTICE help decision support system. It is used when a user is looking for coverage to any claim. The supplement conditional is a model of the contractual clauses of a specific policy. Thus, it is linked to a supplement $\mathcal{P}_i^{r_j,s_k}$. In particular, the system is only interested in the model of the policy coverage, that is to say, in the model of the policy *warranties* (e.g. the warranty against natural risks, its application preconditions, its franchise and its limits). Then, a conditional $\mathcal{C}_i$ can be represented as, $\mathcal{C}_i = \{g_0, \dots g_n\}, \forall x \in [0, n-1]$, each $g_x \equiv (Res, Fra, GrlL, PSinL, PSitL, AgrL)$ containing an applicability precondition $Res \in \mathcal{R}es$ (see section 3.2.3), and several formulas Fra, $GrlL$, $PSinL$, $PSitL$, $AgrL \in \mathcal{F}or$ (see section 3.2.2), used to calculate the franchise and limits (general, per sinister, per RS and aggregated). These expressions can access to the context where they are evaluated and they can use temporal information about RSs, policies and other claims.

3.2.2 Formulas

The set of formulas, together with the RSs meta-description (i.e. RGs) and the set of hazards $\mathcal{H}$, are key elements to build a flexible and adaptable RMIS framework.

Each element $\mathcal{F}or_i \in \mathcal{F}or$ is a formula which has been built using an *ad-hoc* language (which is not within the scope of this paper) and models a receipt, franchise or limits calculation.

$$\mathcal{F}or_i : \{\mathcal{RS}_j^{v_x,r_y} / \mathcal{RS}_j^{v_x,r_y} \in \mathcal{RS}\} \times \mathcal{P}user \times \mathcal{P}sys \longrightarrow \cup Types$$

where $\mathcal{P}user$ and $\mathcal{P}sys$ are sets of pairs label/value which can be accessed using specific constructors of the modelling language. The $\mathcal{P}user$ set, also known as user parameters set, are input values provided by the user in the process of evaluation of a formula. The $\mathcal{P}sys$ set, also known as system parameters, are values calculated by the system, whether it be calculated when the formula is evaluated or through time as internal counters.

3.2.3 Restrictions

Restrictions are the conditions to be hold by a warranty in order to be activated. Every warranty (i.e. contractual clause) inside a conditional of a policy has an applicability restriction which models its behaviour.

Each element $\mathcal{R}es_i \in \mathcal{R}es$ is a restriction which has been build using a language that is a super-set of the formulas language, where logical operators and the concept of *nuance* have been added.

$$\mathcal{R}es_i : \{\mathcal{RS}_j^{v_x,r_y} / \mathcal{RS}_j^{v_x,r_y} \in \mathcal{RS}\} \times \mathcal{P}user \times \mathcal{P}sys \times \{\mathcal{H}_j / \mathcal{H}_j \in \mathcal{H}\} \longrightarrow (Integer, Boolean \cup String)$$

Once a restriction is evaluated, it can be $true$ or $false$ (i.e. the associated warranty can be applied or not). However, sometimes it is not possible to

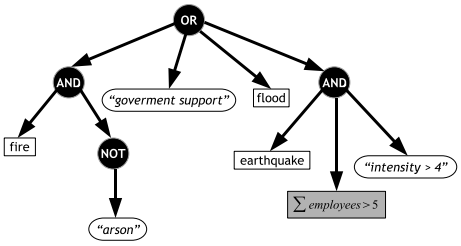

Figure 3: Restriction tree.

state directly whether the restriction is true or not. In this case, the truthfulness of a restriction may depend on the answer of the user to a question expressed in natural language (i.e. a combination of nuances, $String \in Types$). Therefore, it could only be evaluated to a boolean value by means of human user intervention.

The evaluation process of a restriction can be seen, then, as a simplification of the tree which represents the contractual clause: the system, using the context information, removes as many nuances as possible. The result is a tree with a logical combination of nuances that must be evaluated by a human user.

Figure 3 illustrates the appearance of the internal structure of a restriction $\mathcal{R}es_i$. This could be the applicability precondition of a contractual clause.

3.3 Claims Subsystem

The risks and policies subsystems provide us with a full set of tools to model the business framework. The claims subsystem takes that modelling as a basis to carry on its own task: performing registrations of each day's incidents, checking their coverage and tracking their whole life cycle, from opening to closing, managing related tasks depending on the kind of claiming, receipts and relevant documentation, and, of course, remarkable calculations to be done.

Thus, in a formal way, we can describe the set of *claims* in the system, $\mathcal{CM}$, as a set of elements,

$$\begin{aligned}\mathcal{CM}_i \equiv (HDate, \{H_1, \dots H_h\},\\ AffectedRSs, P_i^{r_j,s_k}, g_n)\\ HDate \in Timestamp \wedge\\ AffectedRSs \subseteq \mathcal{RS}\\ \forall RS_{j_1}^{v_{x_1},r_{y_1}}, RS_{j_2}^{v_{x_2},r_{y_2}} / j_1 \neq j_2 \wedge\\ g_n \in (Conditional \in P_i^{r_j,s_k})\end{aligned}$$

that represent the relationship between the RSs affected by one or more hazards on date $HDate$. $HDate$ is also the one used to find the most suitable warranty to cover a claim, that is to say, the one that will be used to locate a supplement with a earlier effective date and a later expiration date. Once the appropriate-by-date $P_i^{r_j,s_k}$ supplements are identified, their warranties must be examined, and the most accurate one (g_n) is chosen by the user to be charged with all the expenses.

As can be seen in this definition, the same comment we made when talking about the policies coverage is applicable here: each $\mathcal{RS}_j^{r_x,s_y}$ inside the $AffectedRSs$ set must be different. As both sets will be checked for total/partial matches, this makes all sense.

4 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a formalisation of the ARMISTICE kernel, pointing out the extensibility and adaptability properties that the model has. We have left out both technical information and low-level details.

Nowadays, ARMISTICE is being successfully deployed in the risk management department of a large holding enterprise for about a year. Its adaptability has been proved as it has been possible to model easily all the insurable elements, policies and hazards. Moreover, the formalism introduced by the need of that modelling has made possible to detect small mistakes in the original composition of the policy clauses. To sum up, the ARMISTICE adaptability and modelling tasks are a differential factor regarding other RMIS systems.

REFERENCES

AON (2005). Aon. http://www.aon.com.

Armstrong, J., Virding, R., Wikström, C., and Williams, M. (1996). *Concurrent Programming in Erlang, Second Edition*. Prentice-Hall.

Cabrero, D., Abalde, C., Varela, C., and Castro, L. (2003). Armistice: An experience developing management software with erlang. *PLI'03. 2nd ACM SIGPLAN Erlang Workshop*, pages 23–28.

Fowler, M. (2005). Design patterns. http://www.martinfowler.com.

Gulías, V. M., Abalde, C., Castro, L., and Varela, C. (2005). A new risk management approach deployed over a client/server distributed functional architecture. *18th International Conference on Systems Engineering (ICSEng'05)*.

Marinescu, F. (2002). *EJB Design Patterns. Advanced Patterns, Processes, and Idioms*. John Wiley & Sons, Inc.

STARS (2005). Stars. http://www.starsinfo.com.

Wadler, P. (1998). Functional programming: An angry half dozen. *SIGPLAN Notices*, 33(2):25–30.