

**A**RMISTICE [1, 2, 3, 4] is a risk management information system (RMIS) designed for a large regional company. How can we evolve its distributed and functional Erlang [5] architecture to a more powerful and reusable one?

## Framework Overview

**O**UR proposal is a layered client/server architecture, based on two well-known architectural patterns: Layers and Model-View-Controller [6].

The *user side* is a lightweight client which only makes remote procedure calls (RPCs), and has no associated logic. The *server side* supports the model and the business logic, and is structured in four tiers.

Source code for implementing these layers can be automatically generated from:

- Client-server communication format **(A)**.
- Use cases **(B)**.
- Domain transfer objects & relationships **(C)**.

## ARMISTICE: Case Study

**A**RMISTICE is an information system devoted to the advanced management of company business-related risks.

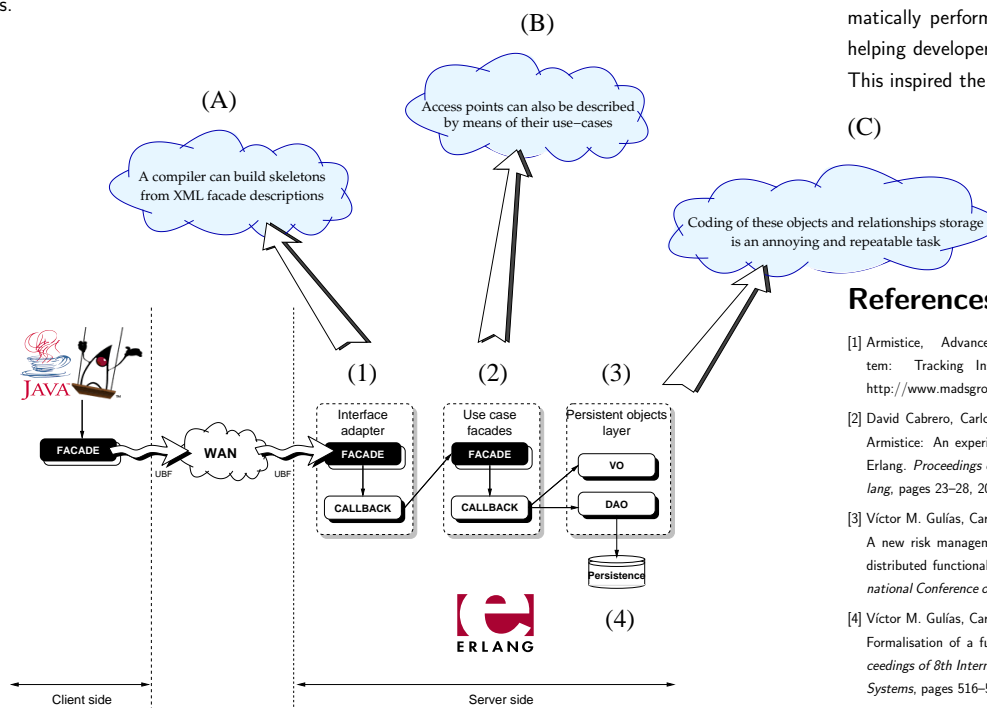
Implementing ARMISTICE server using the functional programming language Erlang:

- ✓ helped to shorten the development cycle
- ✓ allowed its deployment over a low-cost cluster
- ✓ provided scalability
- ✓ gave it fault-tolerance & reliability properties

The use of abstraction (both functional and design patterns) reduces the programming effort to evolve prototypes.

## Conclusions

**B**IG projects such as ARMISTICE, always involve a lot of work which can be automatically performed, saving precious time, and helping developers to focus on non-trivial tasks. This inspired the ideas presented here.



- (1) Interface adapter:** Receives messages from clients and gives them the appropriate format so that the server can process the enquiry.
- (2) Use case facades:** Facades of the system, access points implementing business use cases.
- (3) Persistent objects layer:** Domain entities (TOs) and data access objects (DAOs) [7].
- (4) Persistence:** Permanent storage over a relational database.

Among the supported tasks we can point out:

- Model contracted policies.
- Select the most suitable warranty to cover resources damaged in an accident.
- Manage the claims for accidents.
- Manage another several accident-related tasks (payments, invoices, repairs...).

## References

- [1] Armistice, Advanced Risk Management Information System: Tracking Insurances, Claims and Exposures, 2002. <http://www.madsgroup.org/armistice>.
- [2] David Cabrero, Carlos Abalde, Carlos Varela, and Laura Castro. Armistice: An experience developing management software with Erlang. *Proceedings of the 2003 ACM SIGPLAN workshop on Erlang*, pages 23–28, 2003.
- [3] Víctor M. Gulías, Carlos Abalde, Laura Castro, and Carlos Varela. A new risk management approach deployed over a client/server distributed functional architecture. In *Proceedings of 18th International Conference on Systems Engineering*, pages 370–375, 2005.
- [4] Víctor M. Gulías, Carlos Abalde, Laura Castro, and Carlos Varela. Formalisation of a functional risk management system. In *Proceedings of 8th International Conference on Enterprise Information Systems*, pages 516–519, 2006.
- [5] Joe Armstrong, Robert Virding, Claes Wikström, and Mike Williams. *Concurrent Programming in Erlang, Second Edition*. Prentice-Hall, 1996.
- [6] E. Gamma et al. *Design Patterns. Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1999.
- [7] Floyd Marinescu. *EJB Design Patterns. Advanced Patterns, Processes, and Idioms*. John Wiley & Sons, Inc., 2002.

## Acknowledgements

This work has been partially supported by Spanish MEyC TIN2005-08986 (FARMHANDS: *Recursos Funcionales para la Construcción de Sistemas Distribuidos Complejos de Alta Disponibilidad*).