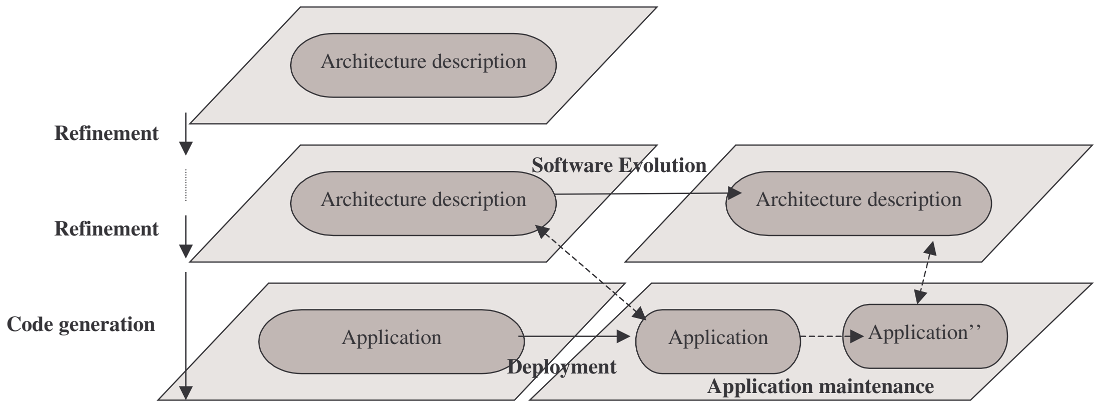
# Challenges in architecture-based software evolution

**Sorana Cîmpan, Herve Verjus**

LISTIC, ESIA Engineering School, University of Savoie
BP 806, 74016 Annecy, Cedex France
Tel: +33(0)4 50.09.65.86 – Fax: +33(0)4 50.09.65.90
Sorana.Cimpan@univ-savoie.fr, Herve.Verjus@univ-savoie.fr

## 1. Architecture centred software process

More and more economic activities rely on software to achieve their business goal, becoming thus software intensive. The change in the economic environment has to be reflected at the level of the software support. Inversely, changes in the technology and software support, induce changes in the economic environment, *i.e.*, the organisation changes in order to better take advantage of the software support. The software applications have to be able to change in order to adapt to the environment evolution (Lehman's law)[13].
During the past years, the work on the engineering of software intensive applications considered the software architecture as a central point in the development process: the architecture is specified early in the software lifecycle, and constitutes the model that drives the entire engineering process.
Thus, first considered at an abstract level, the architecture is stepwise refined till obtaining a concrete representation which sometimes can be used for automatic code generation (see figure below). Software properties can be specified and verified early in the development process, where error recovery costs are lower. A formal refinement guaranties property preservation from one step to another.



***Architecture centred software design and evolution***

Once the application obtained, it follows the classic process: deployment and maintenance. By adopting the architecture centred process, the maintenance phase can benefit from the explicit modelling of the system at different abstraction levels. This explicit representation can be useful especially for the perfective maintenance (which represents 65% of maintenance costs [19]). Moving to higher abstraction levels is interesting if the change impacts the system architecture. This might not be necessary in the case of corrective or adaptive maintenance. *In the remaining of the paper we address by the term* ***software evolution****, the software perfective maintenance, as well as those adaptive and corrective maintenance activities which impact the software architecture.*
In this paper we look on how the software evolution takes place when an architecture centred software development is used, and what are the challenges in this context, with respect to the existing support and approaches for architecture centred development.

## 2. Architecture based software evolution

The adoption of an architecture-centric software development process (cf. figure) induced the proposition of several software development environments which support the architecture centred processes, by: (a) languages to support the definition of software architectures: Architecture Description Languages (ADLs) and associated modelling tools; (b) verification tools (based on model checking, animation, theorem proving, etc.); (c) architecture refiners and code generators (based on rewriting logic, transformation rules). All the proposed ADLs address the software structure, and give means for describing topological aspects of the architecture. Some of the ADLs allow also the representation of the system behaviour [3][8][18][17]. When the software architecture entails behaviour representation, the coupling with the application code is higher.
Reflective ADLs allow passing from one architectural level to code, by proposing reification and reflection mechanisms. In some cases the notation used is the same, at the two levels.

When the deployed software needs to evolve, the reasoning about the evolution is made at the architectural level. So one has to move from the code level to the architectural level.

*Ideally, at every moment the architecture and the code are compliant.*

## 3. Observations and Challenges

***Reflection : the way for architecture centred software evolution***

Making reflective the architecture centred approaches has obvious benefits in keeping the compliance between code and architecture. Several architectural reflection models have been proposed in the literature, which consist in transposing the reflection mechanisms from programming languages to the architectural domain.
Cazzola et al [7] propose the term of architectural reflection, and highlight the benefits gained from using reflective architecture centred processes, one of the main being the broadening of the support from mainly the design phase to run-time management and system evolution. Several meta-levels can be considered, ensuring the code evolution via its architecture, but also the evolution of the architecture via a meta-meta level. Distinction is made between *topological* reflection (associated to structural reflection) and *strategic* reflection (associated to behavioural reflection).
Cuesta et al [9] propose a *M*eta *A*rchitectural *MO*de*L* (*MARMOL*), which equally introduces reflection in the architectural centred process, with a multi-level architectural reflection tower.
Oreizy et al [16] propose to deploy the architectural model as integral part of the application, and use it as bases for evolution. They suggest the use of an Architecture Evolution Manager, which handles the compliance problem by imposing changes to be made at the architectural level and supporting their reflection in the code level[1].
Garlan et al [11] go further, by proposing to make styles first class run-time citizens, and augment them with run-time capabilities to support software adaptation (use the formal constraints to detect anomalies, associate analytical methods that suggest repair strategies, link style constraints with repair rules whose soundness is based on analytical methods, etc.)
The ArchWare project [2] aims are to advance and integrate research on software architecture and reflective systems to develop languages, frameworks and tools for architecting and engineering dynamic and evolvable software systems.

*The reflection is either at the language level, either replaced by specific mechanisms.*

***Reflective ADLs***

Two reflexive ADLs are considered here. Only the second takes into account the code level.

A reflective architecture description language is proposed with the Marmol framework: Pilar [9], based on CCS process algebra extended with reflection specific primitives. Components and meta-components descriptions are

[1] This proposition is part of a framework for architecture-based approach to self-adaptive software.

unified (Pilar is used for modelling both) and passing from base level to meta level. The proposition mainly concerns handling the evolution at the architectural level, and the compliance with the code.

The ArchWare project[2] proposes a reflective architecture description language, ArchWare ADL [17][18][15], allowing the specification of evolvable architectures by a combination of concepts that include: a π-calculus based communication and expression language for specifying executable architectures; hyper-code as an underlying representation of system execution that can be used for introspection; a decomposition operator to incrementally break up executing systems; and structural reflection for creating new components and binding them into running systems [15].

Using an architecture centred development with a reflective ADL implies the adoption of the entire development and enactment environment, as these languages are associated to virtual machines.

The first challenge is that *the adoption has not only to be made by the system developers, but also by the clients*, who will have implicitly the source code and architecture.
Another challenge raised here is that *the languages have to increase their usability and environments have to become more robust and efficient*. With respect to usability, the "implementation related" concepts of the language have the needed expressive power, but have to be completed with end-user library for specific developments. The environment proves the feasibility of reflective architecture driven development, but hasn't passed through large scale system testing.

This approach has definitively to be experienced by other propositions of reflective ADLs. This is clearly a challenge, as all the lifecycle phases have to be covered, the entire development process till maintenance have to be supported.

Efforts for unifying code and architecture representation were also made by ArchJava [1], which is not a reflective ADL, but allows programmers to express architectural structure and then fill in the implementation with Java code. This approach only considers structural aspects and allow to define rather concrete architectures.

***Non-reflective ADLs***

In the case of non-reflective ADLs, we consider the *architectural relevant changes*, which can be expressed at the architectural level, i.e. concern the system architecture. The compliance can be achieved by imposing that all architectural relevant changes have to be done first at the architectural level, and then reflected at the code level (following for instance the architecture centred process: refinement + code generation). Maintenance activities which do not impact the architecture, can take place at the code level, but the code and its architecture remain compliant.

When the ADL employed is not reflective, specific mechanisms have to be introduced to allow one the system reification at the architectural level, make the decision of a change, and reflect the change at the code level. Garlan et al [11] use the feedback loop[3] in order to detect the need for change and implement it by monitoring the running system for compliance to style constraints, decide the need for change and make the change at the architectural level, and then implement it at the code level.
The use of the feedback comes to adopt the reflection, which is not at the language level, but replaced by specific mechanisms (reify by monitoring, reflect by implementing the change).

*While the results are promising, they have to be further experienced to other environments, and generalized.*
Several steps are not formalised enough (the reification and the reflection), and it is difficult to ensure the compliance.

When using non-reflective ADLs, *i.e.*, when code and architecture representations are not unified, guaranteeing compliance by imposing that all architectural level changes have to be done first at the architectural level, is very

[2] The ArchWare proposition was made in the context of a European IST research project [2].
[3] This approach has already been used for other similar problems, such as compliance between software process instances and models [10], in the context of process instance evolutions.

constraining and difficult to achieve. Actually, when making a change it is not always obvious to see if it is architectural relevant or not. Thus architectural relevant changes may take place at the code level only, leading to compliance problems.

## 4. Further considerations

We focused in this paper on the co-evolution between software systems and their software architecture. As mentioned in the introduction, the software support should co-evolve with the organisation for which it offers support. This is one of the key research points stressed out by Bennett and Rajlich in their roadmap on software maintenance and evolution [4], where they talk about further understanding the relationships between technology and business. This co-evolution between software support and business can be tackled also with an architecture centred approach, by representing both the software and the business organisation architectures and reasoning and implementing the change at the architectural level (reflected then in the software system or business organisation). This issue is addressed in the framework of the ArchWare project [6].

## References

[1] J. Aldrich, C. Chambers, D. Notkin, *ArchJava: Connecting Software Architecture to Implementation*, Proceedings of the International Conference on Software Engineering, ICSE'2002, Orlando, Florida, USA, May 2002

[2] ArchWare project team, *Architecting evolvable software, ARCHWARE*, European RTD Project IST-2001-32360, 2001-2004.

[3] R. Allen, R. Douence & D. Garlan, *Specifying and Analyzing Dynamic Software Architectures*. Proceedings on Fundamental Approaches to Software Engineering, Lisbon, Portugal, March 1998.

[4] K.H. Bennett, V. T. Rajlich, *Software maintenance and evolution: a roadmap*, International Conference on Software Engineering, Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, Pages: 73 - 87, 2000

[5] M.Bernardo, P.Ciancarini, L.Donatiello, *Architecting Systems with Process Algebras*. Technical Report UBLCS-2001-7, July 2001.

[6] L. Blanc dit Jolicoeur, R. Dindeleux, A. Montaud, F. Leymonerie, S. Gaspard, C. Braesch. *Definition of Architecture Styles and Process Model for Business Case1,* ARCHWARE European RTD Project IST-2001-32360. Deliverable D7.2b, June 2003.

[7] W. Cazzola, A. Savigni, A. Sosio, F. Tisato. *Architectural Reflection: Bridging the Gap Between a Running System and its Architectural Specification*. In proceedings of 6th Reengineering Forum (REF'98), pp 12-1-12-6, Firenze, Italia, March 1998.

[8] C.Chaudet, F.Oquendo, *π-SPACE: A Formal Architecture Description Language Based on Process Algebra for Evolving Software Systems*, Proceedings of 15th IEEE International Conference on Automated Software Engineering (ASE'00), September 11 - 15, 2000, Grenoble, France.

[9] C.Cuesta, P. de la Fuente, M. Barrio-Solorzano, *Dynamic Coordination Architecture through the use of Reflection*. Proceedings of the 2001 ACM symposium on Applied computing, Las Vegas, Nevada, United States, pp: 134 - 140, 2001

[10] P.Y. Cunin, *The PIE project: An Introduction*, Proceedings of the 7th European Workshop on Software Process Workshop Technology, Reidar Conradi (ed.), LNCS 1780, Kaprun, Austria, February 2000

[11] D. Garlan, S. W. Cheng, B. Schmerl, *Increasing System Dependability through Architecture-based Self-repair*, in Architecting Dependable Systems, R. de Lemos, C. Gacek, A. Romanovsky (Eds), Springer-Verlag, 2003.

[12] M. Greenwood, D. Balasubramaniam, S. Cîmpan, N.C. Kirby, K. Mickan, R. Morrison, F. Oquendo, I. Robertson, W. Seet, R. Snowdon, B. Warboys, E. Zirintsis. *Process Support for Evolving Active Architectures.* Proceedings of the 9th Europeean Workshop on Software Process Technology, EWSPT 2003, Helsinki, Finland, 2003, pp. 112-127.

[13] Lehman MM, *Programs, Life Cycles and Laws of Software Evolution*, Proc IEEE Spec. Issue on Software Engineering, vol. 68, no. 9, pp. 1060 -1076. Sept l980.

[14] R. Milner, J. Parrow, D. Walker, *A Calculus Of Mobile Processes*. Information and Computation, pp 1-40, 1992.

[15] R. Morrison, D. Balasubramaniam, N.C. Kirby, K. Mickan, F. Oquendo, S. Cimpan, B. Warboys, R. Snowdon, M. Greenwood, *Support for Evolving Software Architectures in the ArchWare ADL,* 4th Working IEEE/IFIP Int. Conf. on Software Architecture, WICSA 2004, Oslo, Norway, June 2004, pp. 69-78

[16] Oriezy, P., Gorlick, M.M., Taylor, R.N., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., and Wolf, A. *An Architecture-Based Approach to Self-Adaptive Software*. IEEE Intelligent Systems 14(3):54-62, May/Jun. 1999.

[17] F. Oquendo, *π-ADL: an Architecture Description Language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures*. ACM SIGSOFT Software Engineering Notes Volume 29, Issue 3, May 2004, pp. 1-14

[18] F. Oquendo, I. Alloui, S. Cîmpan, H. Verjus, *The ArchWare ADL: Definition of The Abstract Syntax and Formal Semantics.* ARCHWARE European RTD Project IST-2001-32360, Deliverable D1.1b, 2002.

[19] I. Sommerville, *Software Engineering*, Fifth Edition, Addison-Wesley, 1995.