# BeeEye: A Framework for Constructing Architectural Views

Hervé Verjus[1], Sorana Cîmpan[1], Azadeh Razavizadeh[1], and Stéphane Ducasse[2]

[1] University of Savoie, LISTIC Lab, France
[2] INRIA Lille-Nord Europe, RMoD Team, France
{firstname.lastname}@univ-savoie.fr, stephane.ducasse@inria.fr

**Abstract.** We believe that offering means for defining and building multiple architectural views of a given system enhances the understanding of the system as a whole. BeeEye is a *generic and open framework for architecture reconstruction*, which allows to construct architectural views using different (possibly combined) viewpoints and perspectives. The framework follows a model-driven approach where viewpoints and views (abstract and concrete) are models that are defined, constructed and used.

## 1 Introduction

Software systems need to *evolve* over time. They get modified to improve their performance or change their functionality in response to new requirements, detected bugs, *etc.* Some changes are part of the system maintenance; others evolve the system, generally by adding new functionalities, modifying its architecture, *etc.*. To successfully evolve a complex system, it is essential to *understand* it. The understanding phase is time and effort consuming, due to several reasons, among which: the system size, lack of overall views of the system, its previous (undocumented) evolutions, *etc.* Software architectures are valuable assets during software evolution; they improve the system understanding, by providing abstract representations of it. This motivates us on supporting the software system understanding phases, by constructing for an existing system different architectural representations, at different abstraction levels, called *architectural views*.

It is widely accepted that multiple architectural views are useful when describing the architecture of a software system [11,4,1]. Architecture relevant information can be found at different granularity levels of given systems and needs to be studied from different perspectives. A viewpoint is a collection of patterns and conventions for constructing one type of views. It reflects stakeholders concerns and guides the construction of views [9]. This paper presents the BeeEye framework, dedicated to the construction of architectural views according to different, possibly composed, viewpoints. The framework proposes several viewpoints related to both business and software engineering domains. It also provides the means for defining new viewpoints. The next section presents an overview of the BeeEye approach. Then we zoom on architectural views (Section 3), architectural viewpoints (Section 4), and view construction (Section 5). Section

M. Ali Babar and I. Gorton (Eds.): ECSA 2010, LNCS 6285, pp. 376–383, 2010.

6 presents related work and section 7 briefly addresses the BeeEye framework implementation. The paper closes in section 8 with concluding remarks.

## 2 BeeEye: Goal and Overall Approach

Current propositions highlight the importance of taking into account multiple viewpoints in both the engineering of the system, and in the maintenance phase. Several viewpoints were proposed to be used during the different phases of the software process [11,18,7]. Software architecture recovery aims at extracting architectural representations for existing systems. Ducasse and Pollet [6] propose an exhaustive process-oriented taxonomy of existing architecture reconstruction approaches. Such approaches are classified according to their goal, processes employed, inputs used, techniques and outputs. Given the wide range of propositions, we identified the need for *a unifying architecture recovery framework*, where processes, techniques and views can be combined in different ways, depending on the user expectations. The main constraints on such a framework are: *(1) genericity*: set and structure the main concepts to cover as much as possible the existing techniques; give system representations from different (possible user-defined) perspectives; *(2) flexibility, openness*: provide different construction processes and means to combine them; give the possibility to define user-specific construction processes. None of the existing approaches is generic enough to provide such a framework: they either limit to some specific viewpoints and/or representations, either the process is fixed, either the techniques employed are limited: they are not adapted as a basis for a generic framework, as their intended goal was not to provide such a framework.

BeeEye is a first proposal for a *generic architecture recovery framework*. BeeEye deals with the above mentioned contraints on genericity, flexibility and openness throughout the use of:

- *views*, and *viewpoints*: as the main artefacts for architecture representation and recovery, where views are system representations from a given perspective defined in a viewpoint;
- *composable construction processes*: different basic operators (construction techniques) are provided and can be combined in a flexible manner to obtain user-defined construction processes;
- *different abstraction levels*: architectural representations (views) are considered at different abstraction levels; different kinds of relations exist among constructed views; abstraction and refinement relations concern views situated at different abstraction levels; composition relations concern views situated at the same abstraction level; these relations are inferred either by construction, either by analysis of existing views.

Figure 1 presents an overview of the possibilities offered by the framework in terms of view construction, relations among views and viewpoints. Each construction step corresponds to a framework recursion [16] where an output view is constructed from an input view using a given viewpoint. The viewpoint entails

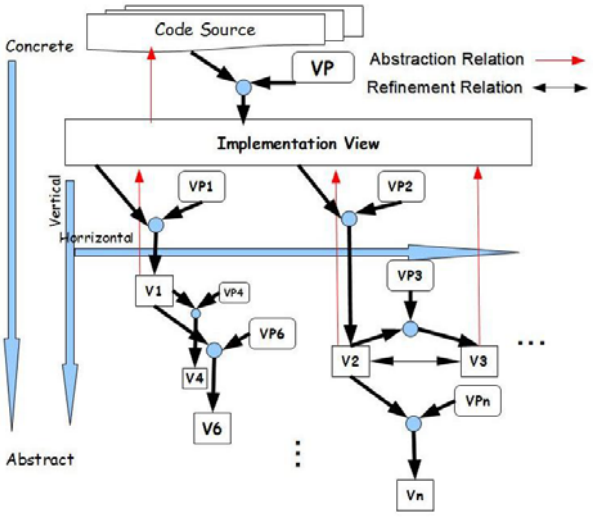


**Fig. 1.** An overview of the BeeEye Framework

the technique used to construct the view. Such construction steps can be chained horizontally and/or vertically. An architectural view construction process can combine multiple construction steps. A vertical application of a construction step leads to output views representing the system from the same perspective as the input view. It induces a change in the abstraction level: the output view is either an abstraction, either a refinement of the input view. Horizontal applications of construction steps lead to composed output views in which the system is represented from multiple perspectives. This translates in a representation which details elements of the input view using a different perspective on the system: the concerned elements are represented as composite elements.

## 3 Architectural Views

An *architectural view* represents a system in terms of interconnected *architectural elements* from a given (possibly composed) perspective. Such a perspective is related to particular stakeholders concerns [9], and conditions the representation elements and their relationships. A view is generally part of a set of views representing the system using different perspectives. Several views may represent the system from the same perspective, but at different abstraction or detail levels. Relations exist among views, generally related to the framework recursions that constitute the view construction process (Section 5). We consider two kinds of architectural views: *abstract views* and *concrete views*.

An *abstract view* represents a possible (intuitive) model for a system within a considered perspective. Each element of the abstract view is supposed to be an abstraction of a part of the system, but the relation with the system's elements is

not explicit. Abstract views are means for representing *a priori* knowledge on the system architecture. Different inputs, architectural or not [6], can be considered when defining abstract views, such as previous architectures, documentation, expertise on the system, *etc.*

A *concrete view* gives a concrete representation for a software system: the relation between this view and the system is explicit, generally by abstraction relationships among view and system elements. Concrete views are considered at different abstraction levels. Elements of a concrete view are either directly, either transitively connected to system elements via abstraction relations. An implementation view is considered for the system, where for each system element a corresponding architectural element is defined and connected by an abstraction relationship. Concrete views are always issued from a construction process (section 5) which employs one or several viewpoints (section 4).

## 4 Viewpoints

[9] defines a viewpoint as a collection of patterns and conventions for constructing one type of view. It reflects stakeholders concerns and guides the construction of views. BeeEye uses viewpoints in each framework recursion where starting from an entry concrete view another concrete view is constructed (see Section 2). BeeEye proposes two general classes of viewpoints: *(a) matching viewpoints*: are used to verify whether the system is compliant (in terms of established criteria) to a given architectural representation (abstract view); the user has thus representation expectations which are tested against the system; *(b) discovery viewpoints*: are used as means for discovering representations of the system in the absence of particular representation expectations; elements are grouped using generic *similarity criteria*.

The two intuitive definitions given above can be refined in terms of *user concerns representation* and *kind of construction process* employed. Thus, user concerns can be represented in terms of *abstract views* (*matching viewpoints*) or/and in terms of a *similarity criteria* to be used and an associated *threshold* (*discovery viewpoints*). In *matching viewpoints* the construction process maps the elements of the system[1] against architectural elements of the abstract view. In *discovery viewpoints* the construction process makes use of clustering techniques; it compares among them elements of the entry concrete view using the chosen similarity criteria (reflecting user concerns). Elements with a degree of similarity above the established threshold are grouped and association relations are defined between them and a corresponding architectural element introduced in the constructed concrete view.

Our framework formalizes thus the [9] definition by separating the concerns reflected in the viewpoint (in an abstract view for mapping viewpoints and similarity criteria for discovery viewpoints) on the one hand, and rather generic construction primitives that make use of this information when constructing a

[1] We use the term system elements here to make reference to elements of the view used as an entry view for the construction process.

new view on the other hand. The separation of the viewpoint definition in these distinct, yet related descriptions, has several benefits: *(i) reusability* and *maintenance flexibility*: each part of the framework (abstract and concrete views, viewpoints, construction process) can be maintained and reused independently; *(ii) accessibility* and *security*: this separation gives the ability to use the framework for different categories of users with different levels of knowledge about a system.

Examples of matching viewpoints are *business domain*-based mapping viewpoints which consider the principal business domain concepts and their relationships, and *software pattern*-based viewpoints [8] which identify architectural elements conform to a given pattern. Examples of discovery viewpoints are the *activity*-based viewpoint which identifies the architectural elements according to their level of interaction with their environment, and the *business domain*-based discovery viewpoint which identifies business domain concepts. The framework can easily integrate other viewpoints.

## 5 View Construction

Concrete views are always constructed using framework recursions, or construction steps 1. Such a step takes an existing concrete view as input and produces another concrete view using a viewpoint. The specificity of concrete views relies in their relations with the system they represent (dashed arrows in Figure 1). They provide abstract representations of the system, from a given perspective entailed implicitly in the viewpoint definition. This relation can be direct, if the view was constructed directly from the system. Otherwise, it can be obtained by transitive closure, as each concrete view posses relations towards the view from which it was constructed. So at each framework recursion the view constructed is linked to the input view. The nature of this relation depends on the technique employed: construction by *refinement*, *composition* or *abstraction*.

Construction steps can be chained, combining vertical and horizontal recursions, and concrete views are issued from a succession of construction steps. We employ the term *construction process* to make reference to this combination of construction steps. As each step employs a different viewpoint each of which can be related to a different perspective, a view can represent the system from a combined perspective.

*Construction by Abstraction.* The elements of the constructed view are at a higher abstraction level than their related elements of view given as input. Elements of the input view sharing a particular characteristic are grouped. Characteristics used for grouping elements are either provided by the abstract view (matching viewpoint), either they are provided by generic algorithms ( *i.e.* detecting elements' naming similarities - discovery viewpoint). The elements of the constructed view have abstraction relationships towards elements of view given as input.

*Construction by Refinement.* This technique is the counterpart of the previous one. The output view represents the system from the same perspective, but at a lower abstraction level. It consists a more detailed representation of the system.

*Construction by Composition.* This technique is employed to obtain multiple perspective views and corresponds to horizontal framework recursions. The viewpoint $V$ employed in a construction by composition step corresponds to a perspective that differs from the one in which the input view represents the system. Thus, for each element $E$in the entry view, the associated abstracted elements are considered and grouped according to the viewpoint $V$. The elements thus obtained and their relationships are considered as a representation of the element $E$ and bare composition relationships to it.

## 6 Related Work

Various contributions concern architecture recovery for object-oriented systems [6]. The inputs used by extraction approaches are various. Most often the source code is used, but also alternative sources of information such as: developer knowledge [13,10]; bug reports and external documentation [2]; or an ontology of the software system's domain [3]. In our approach we use viewpoints to guide the extraction from the source code of a system. Viewpoints are generic and can be related to a software pattern, a business model or cohesion metric,*etc.*. Separating user concerns and construction process in viewpoint definition increases their genericity, reuse and maintainance.

There are several techniques to reconstruct architecture of an existing system. Approaches like [12] and [15] consider external constraints (represented as queries) to be checked against the reality of source code or recovered architectural elements. [13,10,17] propose an automatic reconstruction technique based on reflexion models, starting with a structural high-level model. In Murphy et al. proposition, users iteratively refine a structural high level view model to gain information about the source code. The technique is based on the definition of a set of mappings between this high level model and the source code. Our technique is a reflexion model; the main difference is that we propose a framework to apply this reflexivity. This framework leads in define multiple views from any generated (or existing) view.

## 7 Implementation

Conceptually, the BeeEye framework entails all reconstruction steps starting from the source code. Nevertheless, the initial steps correspond to reverse engineer the system. The BeeEye implementation (in Smalltalk) uses the Moose re-engineering environment [14] for the construction of the implementation view (the first BeeEye framework recursion) which is represented using the FAMIX meta-model [5]. The current framework implementation supports part of the proposed techniques. Although not complete, the implementation allowed us to test both vertical and horizontal view construction. The paper [16] details and further analyses the results obtained in a case study using the BeeEye framework.

## 8 Concluding Remarks

We propose in this paper BeeEye, a *generic architecture recovery framework*. The architecture is defined here as a set of *architectural views* representing the system from different perspectives and at different abstraction levels. In building this framework we tried as much as possible to cover existing propositions and to build an open framework that eases the integration of new means for view construction. Thus, BeeEye provides generic enough concepts to cover as much as possible the existing extraction techniques and to support system representations from different (possible user-define) perspectives. It equally provides different construction processes and means to combine them, giving the possibility to define user-specific construction processes. None of the existing approaches in software architecture recovery is generic enough to provide an open and generic framework for architecture recovery: they either limit to some specific representations, either the extraction process is fixed, either the techniques employed. Using the BeeEye framework, viewpoints related to both business and software engineering domains are defined. It also provides means for defining new viewpoints.

## References

1. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond. Addison-Wesley Professional, Reading (2002)
2. Cubranic, D., Murphy, G.: Hipikat: Recommending pertinent software development artifacts. In: Proceedings 25th International Conference on Software Engineering (ICSE 2003), pp. 408–418. ACM Press, New York (2003)
3. Deissenboeck, F., Ratiu, D.: A unified meta-model for concept-based reverse engineering. In: Proceedings of the 3rd International Workshop on Metamodels, Schemas, Grammars and Ontologies, ATEM 2006 (2006)
4. Deursen, A., Hofmeister, C., Koschke, R., Moonen, L., Riva, C.: Symphony: View-driven software architecture reconstruction. In: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 122–134 (2004),
`http://csdl.computer.org/comp/proceedings/wicsa/2004/2172/00/21720122abs.htm`
5. Ducasse, S., Gîrba, T., Greevy, O., Lanza, M., Nierstrasz, O.: Workshop on FAMIX and Moose in software reengineering (FAMOOSr 2008). In: 15th Working Conference on Software Maintenance and Reengineering (WCRE 2008), October 2008, pp. 343–344 (2008),
`http://scg.unibe.ch/archive/papers/Duca08bFAMOOSr2008.pdf`
6. Ducasse, S., Pollet, D.: Software architecture reconstruction: A process-oriented taxonomy. IEEE Transactions on Software Engineering (2009),
`http://scg.unibe.ch/archive/external/Duca09x-SOAArchitectureExtraction.pdf`
7. Finkelstein, A., Goedicke, M., Karmer, J., Niskier, C.: Viewpoint oriented software development: Methods and viewpoints in requirements engineering. In: Algebraic Methods II: Theory, Tools and Applications (1991)

8. Guo, Y., Atlee, Kazman: A software architecture reconstruction method. In: Working Conference on Software Architecture (WICSA), pp. 15–34 (1999)
9. IEEE Architecture Working Group: IEEE P1471/D5.0 Information Technology — Draft Recommended Practice for Architecural Description (August 1999)
10. Koschke, R., Simon, D.: Hierarchical reflexion models. In: Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003), p. 36. IEEE Computer Society, Los Alamitos (2003)
11. Kruchten, P.B.: The 4+1 view model of architecture. IEEE Software 12(6), 42–50 (1995)
12. Mens, K., Kellens, A., Pluquet, F., Wuyts, R.: Co-evolving code and design with intensional views — a case study. Journal of Computer Languages, Systems and Structures 32(2), 140–156 (2006), http://prog.vub.ac.be/Publications/2005/vub-prog-tr-05-26.pdf
13. Murphy, G., Notkin, D., Sullivan, K.: Software reflexion models: Bridging the gap between source and high-level models. In: Proceedings of SIGSOFT 1995, Third ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 18–28. ACM Press, New York (1995)
14. Nierstrasz, O., Ducasse, S., Gîrba, T.: The story of Moose: an agile reengineering environment. In: Proceedings of the European Software Engineering Conference (ESEC/FSE 2005), pp. 1–10. ACM Press, New York NY (2005), (invited paper) http://scg.unibe.ch/archive/papers/Nier05cStoryOfMoose.pdf
15. Pinzger, M., Fischer, M., Gall, H., Jazayeri, M.: Revealer: A lexical pattern matcher for architecture recovery. In: Proceedings of the 9th Working Conference on Reverse Engineering (WCRE 2002), pp. 170–178 (2002)
16. Razavizadeh, A., Cîmpan, S., Verjus, H., Ducasse, S.: Software system understanding via architectural views extraction according to multiple viewpoints. In: 8th International Workshop on System/Software Architectures, Algarve, Portugal (November 2009)
17. Robillard, M.P., Murphy, G.C.: Concern graphs: finding and describing concerns using structural program dependencies. In: ICSE 2002: Proceedings of the 24th International Conference on Software Engineering, pp. 406–416. ACM Press, New York (2002)
18. Woods, S.G., Carrière, S.J., Kazman, R.: The perils and joys of reconstructing architectures. SEI Interactive, The Architect 2 (September 1999)