Views for the Integration of Datatypes and Control Specifications (Extended Abstract)

Christine Choppy[1], Pascal Poizat[2], and Jean-Claude Royer[2]

[1] LIPN, Institut Galilée - Université Paris XIII,
Avenue Jean-Baptiste Clément, F-93430 Villetaneuse, France
Christine.Choppy@lipn.univ-paris13.fr
[2] IRIN, Université de Nantes
2 rue de la Houssinière, B.P. 92208, F-44322 Nantes cedex 3, France
{Pascal.Poizat,Jean-Claude.Royer}@irin.univ-nantes.fr
http://www.sciences.univ-nantes.fr/info/perso/permanents/poizat/

1 Introduction: on Formalism Requirements

We herein deal with mixed specification formalisms, *i.e.* formalisms dealing both with the static (data types) and the dynamic (behaviour) parts of the systems such as LOTOS [23], SDL [15], Estelle [22], UML [37], Raise [38] or extension of non mixed formalisms as ObjectZ/CSP [36], CCS [28], CSP [21]. Moreover, we agree with [34] who think object-oriented concepts (reuse, encapsulation, inheritance for example) are as important for specification as they are for programming. Therefore, our work takes into account three main concerns: Concurrency - or control, communication, behaviour - , Data types and Object-orientation - and object-basis. The formalism is intended to be used at a design level (from the end of the analysis that may be done in UML for example to the code generation).

Defining or choosing a new specification formalism [7], one should take care to keep a good balance between readability and user-friendliness, expressiveness and abstraction.

Notations : a Deal Between Expressiveness and Readability Full specifications cannot be fully graphical for expressiveness sake but a formalism should contain accepted graphical notation whenever possible [32, 12]. A dual system with a graphical notation (with some textual components, used for example in composition) and an associated textual counterpart seems to be the best approach. Fully graphical (UML [37] without OCL) or fully textual (CCS [28], CSP [21], Π-calculus [27]) are not practical either at a formal level (a neat graphical notation with no formal semantics hence no verification) or at readability when dealing with real size systems. Some other systems use both notations (SDL [15], Raise [38], Argos [25], CPN [24]), some other enable the "derivation" of a graphical representation from the (or part of the) textual specification (Estelle [22], LOTOS [23]).

Compositionality A semantics is said to be compositional when the semantics of composed objects can be derived from the semantics of the components they are made from. Compositionality allows modular specification (hence increases their readability), and advocates to give subcomponents semantics in terms of interfaces. Details about components behind their interfaces are not relevant so compositionality is closely related to full abstraction. Examples of formalisms with compositional semantics are LOTOS [23] or Argos [26].

(Full) Abstraction A semantics is fully abstract if it contains no unnecessary (implementation) details. Fully abstract specified components are given in terms of their external behaviour (their interface) and components with equal external behaviours (they behave in the same way in the (any) same context) will be considered as equivalent. Full abstraction requires specific equivalence or congruence relations (see observational equivalence for example). Component compatibility and subtyping may then be defined on the basis of this fully abstract equivalence notion [35].

Communication Semantics As we focus on mixed specifications, expressiveness refers not only to data types but also to the dynamic behaviour of specified components and to their relations. There are different possibilities: synchronous or asynchronous events, events interleaving (or not), synchronous or asynchronous communications, logical or physical time, ... Formalisms dealing with the dynamic part of systems are based upon different combinations that build their communication semantics. A classification of different models of concurrency and the relationships they share is given in [33, 29].

Concurrent Object Orientation As we focus on mixed specification formalisms, and think that reusability is a key word for both specification and programmation production, code generation will naturally be targeted at concurrent object oriented languages [1, 9]. Surveys on COO programming languages are given in [19, 8], comparison of OO formal methods in [18, 2].

To reduce gaps between abstract specifications and implementation levels, the formalism and the language targeted by the code generation should have as much corresponding concepts as possible [3, 12, 13, 31], and notions of class/instances, object/processes identifiers, inheritance/subtyping should be present in the formalism. OMTroll [13], Statecharts [20] or Object Oriented Petri Nets [6, 5, 18] are examples of such formalisms that include (some) object oriented concepts in a graphical formalism.

2 Views for the Integration

We present here the integration means provided by our formalism, called KORRIGAN[1]. The whole formalism is described in [11, 10].

[1] Korrigan: from the breton *korrig* (pixy, dwarf), and *gan* (to sing). Singing pixy.

2.1 The View Model

We define an *Internal Structuring View* abstraction (Fig. 1) that expresses the fact that, in order to design an object, it is useful to be able to express it under its different aspects (here the static and dynamic aspects, with no exclusion of further aspects that may be identified later on).

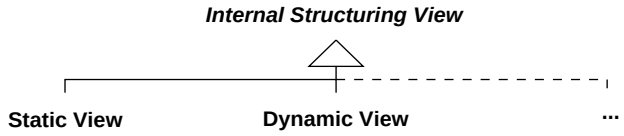

Fig. 1. Internal Structuring class diagram for the view model

We therefore have here a first structuring level (*internal structuring*) for objects. Another structuring level is the *external structuring* (Fig. 2), expressed by the fact that an object may be composed of several other objects. An object may then be either a simple one (integrating different internal structuring views in an *integration view*), or a composite object (*composition view*).

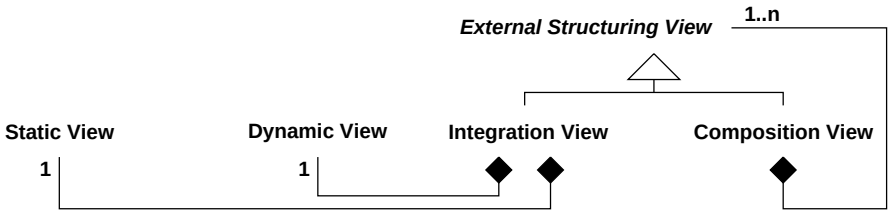

Fig. 2. External Structuring class diagram for the view model

Integration views follow an *encapsulation principle*: the static part (data type) may only be accessed through the dynamic part (behaviour) and its identifier (`Id`).

The whole class diagram for the view model is given in Figure 3.

2.2 Integration Means

An external structuring view (ESV) may contain one or more global semantics components (*i.e.* integration views or composition views). External structuring views have a global semantics.

Definition 1 (External Structuring View). *An* external structuring view ESV_T *of a component* T *is a triple* (A_T, Θ, K_T) *where* A_T *is an optional data part,* Θ *is the ESV "glue" part, and* K_T *the ESV composition layer.*

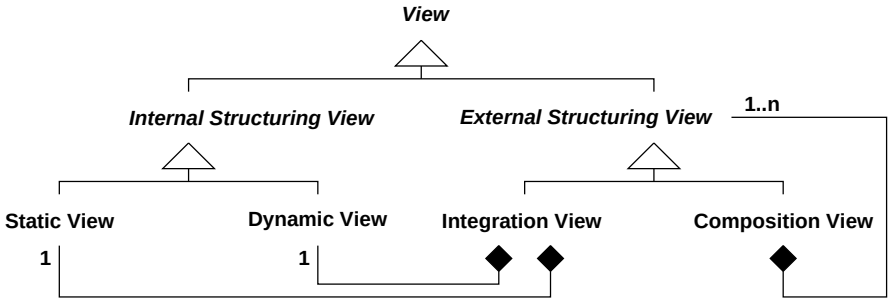

Fig. 3. The View model class diagram

Some new parameters and/or basic algebraic data types may be needed in the ESV.

Definition 2 (ESV data part). *An ESV* data part A_T *(in an ESV structure* (A_T, Θ, K_T)*) is made up of following components* $(A', G, V, \overline{A})$ *where:*

- A' *is a set of imported basic algebraic specifications (datatypes without views)*
- G *is a set of formal parameter algebraic specifications*
- V *are parameter variables*
- $\overline{A}$ *are hidden (or non-exported) operations.*

Θ describes [10] how the components of T are glued altogether.

Definition 3 (ESV glue part). *An ESV* glue part Θ *(in an ESV structure* (A_T, Θ, K_T)*) is a tuple* $(Ax_\Theta, \Phi, \Psi, \Phi_0, \delta)$ *where:*

- Ax_Θ *are axioms relating guards and operations across different views*
- Φ *is a (state) formula built over id-indexed* C *members of the components views*[2]*.*
- Ψ *is a (transition) id-indexed formula relating (gluing) transitions from several components STS. This is a generalized form of synchronized product vector [4].*
- Φ_0 *is a (state) id-indexed formula expressing the initial states of the components (it should be coherent with the* $\Phi_0 s$ *of the composition components, that is* $\vdash \Phi_0 \Rightarrow \bigwedge_{id} id.\Phi_0(Obj_{id})$*).*
- δ *defines what to do with non-glued transitions (see [10]).*

Φ and Ψ are implicitly universally quantified over time (*i.e.* we have[3] $AG\Phi$ and $AG\Psi$). Ψ is given as couples $\Psi = \{\psi_i = (\psi_{i_s}, \psi_{i_d})\}$, and means $AG \bigwedge_i \psi_{i_s} \Leftrightarrow \psi_{i_d}$.

There are three options for δ: **LOOSE** (non-glued transitions are lost), **ALONE** (non-glued transitions may happen alone, other components do some ϵ transition), and **KEEP** (non-glued transitions may happen alone or with some others non-glued transitions of the other components).

[2] This is used for example to express (mutual) exclusion between two conditions/states of two different components.

[3] Where AG denotes "always globally".

EXTERNAL STRUCTURING VIEW T	
SPECIFICATION	**COMPOSITION** δ
imports A' **generic on** G **variables** V **hides** $\overline{A}$	**is** $id_i : Obj_i[I_i]$ **axioms** Ax_Θ **with** Φ, Ψ **initially** Φ_0

Fig. 4. The KORRIGAN syntax (external structuring views)

INTEGRATION VIEW T	
SPECIFICATION	**COMPOSITION** δ
imports A' **generic on** G **variables** V **hides** $\overline{A}$	**is** STATIC s: $Obj_s[I_s]$ DYNAMIC d: $Obj_d[I_d]$ **axioms** Ax_Θ **with** Φ, Ψ **initially** Φ_0

Fig. 5. The KORRIGAN syntax (integration views)

Definition 4 (ESV composition layer). *An ESV* composition layer K_T *(in a composition view structure* (A_T, Θ, K_T)*) is a tuple* (Id, Obj_{id}, I_{id}) *where:*

- Id *is an identifier of sort* PId_T*.*
- Obj_{id} *is a id-indexed family of objects (integration views or composition views). Identifiers follow the equality and structuring principles.*
- I_{id} *are id-indexed sets of terms instantiating the components parameters.*

The syntax for external structuring views in KORRIGAN is given in Figure 4.

Integration views. Integration views (IV) are used to describe objects that have a global semantics (*i.e.* the integration of all the object aspects). Integration views are a special kind of external structuring views (same structure) with the constraint that it is composed of exactly one view per aspect of the component.

The syntax for integration views in KORRIGAN is given in Figure 5.

Composition views. A composition view (CV) is an external structuring view that may contain one or more global semantics components (*i.e.* integration views or composition views). A composition corresponds to the union of sub-components. Its structure and the corresponding KORRIGAN syntax is the same as for external structuring views.

3 Conclusion

A "specification language" is a compound of at least three parts: the syntax, a method and a semantics. We have sketched our underlying model based on the integration of data types and control by the way of views. In our unifying

approach, views are used to describe the interface for data types and dynamic parts (control and communications). Structuring mechanisms are provided by the way of internal views (or simple view) and external views (composite views). A glue part is used to express interactions between the different components of the external views.

The design of our model is motivated by some general requirements but also from experimentations in the specification of case studies. We proposed a method [30] for the specification of mixed components (*e.g.* LOTOS or SDL) based on the separation of the static and dynamic parts. These methods refine existing ones with a preliminary step of extracting conditions. Thereby a state/transition system is built and this system helps the generation of the data types.

The last question is about semantics. We proposed a first approach with an operational semantics in [10]. The main difficulty is to express the global semantics of an external view which contains several components. An ESV describes how its components are glued altogether using state and transition formulas. We defined logical rules for a state and a transition to verify a formula. The semantics of an ESV is done in two steps: to get a view structure from an ESV and to give a semantics for complex transitions. An event rewrites the system state (a global state, a data type term and an environment) into another system state. Let us mention the approach of [16] that presents a specification technique based on graph transformations used to construct the automatic integration of views.

Classification. In [14], the authors propose an integration paradigm for specifications. Its aim is to deal with the integration of specifications for the different aspects of systems. Hence they propose a four layer integration categorization of integration methods and formal languages. The first layer deals with data types (types and operations). The second layer deals with the data type seen as a set of states and transformations. The third layer deals with what they call the process aspects: the components are seen as concurrent communicating entities. Finally, the fourth layer deals with the system architecture (structuring means).

The differences with our work are:

- We already have a state transition vision of our data types (the STS part of static views, and the constructive axiomatization that is guided by this STS [30]). Hence, we aggregate the first two layers as "static views". Moreover, our views are somewhat abstracted. The second layer of [14] may correspond to the usual "tuples of values" vision of systems. It is well known that this may lead to state explosion problems.
- We do not have a layered approach. We advocate the "separation of concerns" approach (for example to avoid the inheritance anomaly). Static and dynamic views of systems must be able to be used (and reused) separately. Therefore, our work is maybe closer to [17] approach.
- We use a unified formalism (views) for both "layers". This makes the integration somewhat easier.

But, we still may use the [14] categorization in layers to present our formalism [11, 10]:

Layer 1. basic algebraic specifications.

Layer 2. in conjunction with the first layer, static views.

Layer 3. corresponds to their high level case. Dynamic views, synchronous communication mechanisms.

Layer 4. basic structuring means. Both internal (Integration Views, no layers, views are tied using a temporal logic glue) and external (Composition Views). Use of union of views and indexing.

References

1. Luis A. Álvarez, Juan M. Murillo, Fernando Sánchez, and Juan Hernández. Active-Java, un modelo de programación concurrente orientado a objeto. In *III Jornadas de Ingenería del Software, Murcia, Spain*, 1998.
2. Pascal André. *Méthodes formelles et à objets pour le développement du logiciel : Etudes et propositions.* PhD Thesis, Université de Rennes I (Institut de Recherche en Informatique de Nantes), Juillet 1995.
3. Pascal André, Dan Chiorean, and Jean-Claude Royer. The formal class model. In *Joint Modular Languages Conference*, pages 59–78, Ulm, Germany, 1994. GI, SIG and BCS.
4. André Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Etudes et recherches en informatique. Masson, 1992.
5. Eugenio Battiston, Alfredo Chizzoni, and Fiorella De Cindio. Inheritance and Concurrency in CLOWN. In *1st Workshop on Object-Oriented Programming and Models of Concurrency*, Turin, Italy, 1995.
6. Olivier Biberstein and Didier Buchs. Structured Algebraic Nets with Object-Orientation. In G. Agha and F. De Cindio, editors, *Workshop on Object-Oriented Programming and Models of Concurrency'95*, pages 131–145, 1995.
7. Paul E. Black, Kelly M. Hall, Michael D. Jones, Trent N. Larson, and Phillip J. Windley. A Brief Introduction to Formal Methods. In *Proceedings of the IEEE 1996 Custom Integrated Circuits Conference*, 1996.
8. Jean-Pierre Briot and Rachid Guerraoui. Objets pour la programmation parallèle et répartie : intérêts, évolutions et tendances. *Technique et science informatiques*, 15(6):765–800, 1996.
9. D. Caromel and J. Vayssière. A Java Framework for Seamless Sequential, Multi-threaded, and Distributed Programming. In *ACM Workshop "Java for High-Performance Network Computing"*, pages 141–150, Stanford University, Palo Alto, California, 1998.
10. Christine Choppy, Pascal Poizat, and Jean-Claude Royer. A Global Semantics for Views. Rapport de Recherche 189, IRIN, 1999. /papers/rr189.ps.gz in Poizat's web page.
11. Christine Choppy, Pascal Poizat, and Jean-Claude Royer. Control and Datatypes using the View Formalism. Rapport de Recherche 188, IRIN, 1999. /papers/rr188.ps.gz in Poizat's web page.
12. Eva Coscia and Gianna Reggio. JTN: A Java-Targeted Graphic Formal Notation for Reactive and Concurrent Systems. In Jean-Pierre Finance, editor, *Fundamental Approaches to Software Engineering (FASE'99)*, volume 1577 of *Lecture Notes in Computer Science*, pages 77–97. Springer-Verlag, 1999.

13. G. Denker and P. Hartel. TROLL – An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics. Informatik-Bericht 97–03, Technische Universität Braunschweig, 1997. Available at http://www.cs.tu-bs.de/idb/html_e/home/denker/pub_97.html.
14. Hartmut Ehrig and Fernando Orejas. Integration and Classification of Data Type and Process Specification Techniques. Technical report, TU Berlin, July 1998.
15. Jan Ellsberger, Dieter Hogrefe, and Amardeo Sarma. *SDL : Formal Object-oriented Language for Communicating Systems*. Prentice-Hall, 1997.
16. Gregor Engels, Reiko Heckel, Gabriele Taentzer, and Hartmut Ehrig. A combined reference model and view-based approach to system specification. *International Journal of Software Engineering and Knowledge Engineering*, 7(4):457–477, 1997.
17. Robert Geisler, Marcus Klar, and Claudia Pons. Dimensions and Dichotomy in Metamodeling. In *Northern Formal Method Workshop 98 (NFMW'98)*. BCS, Springer, 1998.
18. N. Guelfi, O. Biberstein, D. Buchs, E. Canver, M-C. Gaudel, F. von Henke, and D. Schwier. Comparison of object-oriented formal methods. Technical report of the esprit long term research project 20072 "design for validation", University of Newcastle Upon Tyne, Department of Computing Science, 1997. ftp://lglftp.epfl.ch/pub/Papers/guelfi_coofm.ps.gz.
19. Rachid Guerraoui. Les langages concurrents à objets. *Technique et science informatiques*, 14(8):945–971, Octobre 1995.
20. David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.
21. C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August 1978.
22. ISO/IEC. ESTELLE: A Formal Description Technique based on an Extended State Transition Model. ISO/IEC 9074, International Organization for Standardization, 1989.
23. ISO/IEC. LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. ISO/IEC 8807, International Organization for Standardization, 1989.
24. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Monographs in Theoretical Computer Science. Springer-Verlag, 1997. 3 volumes.
25. M. Jourdan and F. Maraninchi. A modular state/transition approach for programming real-time systems. In *ACM Sigplan Workshop on Language, compiler and tool support for real-time systems*, Orlando, FL, June 1994.
26. F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. In *CONCUR*, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.
27. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, (100):1–77, 1992.
28. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.
29. Mogens Nielsen, Vladimiro Sassone, and Glynn Winskel. Relationships between Models of Concurrency. In *REX'93 : A Decade of Concurrency*, volume 803 of *Lecture Notes in Computer Science*, pages 425–475. Springer-Verlag, 1994.
30. Pascal Poizat, Christine Choppy, and Jean-Claude Royer. Concurrency and Data Types: A Specification Method. An Example with LOTOS. In J. Fiadeiro, editor,

Recent Trends in Algebraic Development Techniques, Selected Papers of the 13th International Workshop on Algebraic Development Techniques WADT'98, volume 1589 of *Lecture Notes in Computer Science*, pages 276–291, Lisbon, Portugal, 1999. Springer-Verlag.

31. J. Ramos and A. Sernadas. A Brief Introduction to GNOME. Research report, Section of Computer Science, Department of Mathematics, Instituto Superior Técnico, 1096 Lisboa, Portugal, 1995. Available at ftp://ftp.cs.math.ist.utl.pt/pub/SernadasA/95-RS-GnomeInt.ps.
32. Gianna Reggio and Mauro Larosa. A graphic notation for formal specifications of dynamic systems. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 40–61. Springer-Verlag, September 1997. ISBN 3-540-63533-5.
33. Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. A Classification of Models for Concurrency. In *Proceedings of Fourth International Conference on Concurrency Theory, CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 82–96. Springer-Verlag, 1993.
34. Amílcar Sernadas, Christina Sernadas, and José Felix Costa. Object Specification Logic. *Journal of Logic and Computation*, 5(5):603–630, 1995.
35. Graeme Smith. A Fully-Abstract Semantics of Classes for Object-Z. *Formal Aspects of Computing*, 7(E):30–65, 1995.
36. Graeme Smith. A Semantic Integration of Object-Z and CSP for the Specification of Concurrent Systems. In J. Fitzgerald, C.B. Jones, and P. Lucas, editors, *Formal Methods Europe (FME'97)*, volume 1313 of *Lecture Notes in Computer Science*, pages 62–81. Springer-Verlag, 1997.
37. Rational Software and al. Unified Modeling Language, Version 1.1. Technical report, Rational Software Corp, http://www.rational.com/uml, September 1997.
38. The Raise Method Group. *The RAISE Development Method*. The Practitioner Series. Prentice-Hall, 1995.