HAL
open science

Service Oriented Architecture Definition Using Composition of Business-Driven Fragments

Sébastien Mosser, Mireille Blay-Fornarino, Michel Riveill

To cite this version:

Sébastien Mosser, Mireille Blay-Fornarino, Michel Riveill. Service Oriented Architecture Definition Using Composition of Business-Driven Fragments. Models and Evolution(MODSE'09), MODELS'09 workshop, Oct 2009, Denver, Colorado, United States. pp.1-10. hal-00531039

HAL Id: hal-00531039

https://hal.science/hal-00531039v1

Submitted on 1 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Service Oriented Architecture Definition Using Composition of Business–Driven Fragments

Sébastien Mosser, Mireille Blay–Fornarino, and Michel Riveill

University of Nice – Sophia Antipolis
Cnrs, I3s Laboratory, Modalis & Rainbow Teams
Sophia Antipolis, France
{mosser,blay,riveill}@polytech.unice.fr

Abstract. Services Oriented Architecture are built through the composition of services (e.g. Web Services) to define complex business process (e.g. Orchestrations). Well known methodologies focus on identifying services and orchestrations at design time. However the orchestration design phase is still a heavy burden, as it induces to deal with both technical and business domain concerns. This article proposes to use an *evolution* framework (Adore) to capitalize architects knowledge and best practices into "evolutions". Architects can build business-driven orchestrations by composing reusable "evolutions" following a *design–by–composition* approach. We apply this approach to build a legacy Soa called Seduite (validation platform for the French national research project Faros).

1 Introduction

Web Services and Orchestrations [1] provide a way to implement scalable Soa (Services Oriented Architecture, [2]) under the Wsoa acronym (Web Services Oriented Architecture). Architects define Web Services to publish elementary services. Complex business processes are defined through orchestrations which assemble elementary functionalities into a control–flow. The W3c defines orchestrations as "*the pattern of interactions that a Web Service agent must follow in order to achieve its goal*" [3]. Orchestrations are the keystones of a Wsoa, as they represent the business-driven processes. Existing methodologies (*e.g.* Soma [4]) focus on process identification. When using these methods, architects identify business services and orchestrations at the model level. Existing high–level formalisms and languages are used to express these processes (*e.g.* Bpmn [5], Bpel [6]). But the process definition must be written from scratch as code. A business–driven *step-wise* development implies to know what happened in previous steps before modifying the legacy software in a coherent way: when defining different solutions corresponding to different customers of a same business domain, each architect will then use its own approach to define solutions which are then difficult to compare and even more to make evolve.

The contribution of this paper is to show how a business architect can design a Wsoa by composing reusable process fragments. It proposes a *business* point of view of the software evolution problematic, focusing on high level evolution

expression. Performing an evolution in our context means enhancing an existing business process, adapting it to fit customers expectations. We are positioning our work at design time: an architects wants to design a business–process, so he/she will retrieve a design model of an existing business–process and adapt it to his/her customer needs.

Where Aspect Oriented Programming (AOP, see sec. 5) targets on reusable aspect identification and weaving, this work focuses on the composition of evolution at the model level and is presented as a complementary approach. We use as running example (described in sec. 2) a reference WSOA called SEDUITE. Section 3 describes the ADORE evolution framework and its benefits for SEDUITE case study. Section 4 exposes validation of this work on the whole legacy system, and section 5 discusses related work. Finally, section 6 concludes this paper by showing some exciting perspectives of this work.

2 Running Example: the SEDUITE system

SEDUITE is an information system designed to fit academic institution needs. It supports information broadcasting from academic partners (*e.g.* transport network, school restaurant) to several devices (*e.g.* user's smart-phone, PDA, desktop, public screen). This system is built upon a WSOA and used as a validation platform by the FAROS project[1]. The implementation follows WSOA methodological guidelines [7], positioning experimentations as a typical usage of a WSOA. The system is deployed inside two institutions. Further information about implementation and partners can be found on the project web site[2].

The SEDUITE system defines two kinds of business processes: *(i) sources* of information to retrieve information from partners and *(ii) providers* to assemble sources invocations according to broadcast policies. Designing such processes is a business–driven task, as they must fit with school users needs. We identify inside the SEDUITE system a set of chronic situations where the business architect reuse a set of *good practices* inside system's orchestrations. We present in this section a relevant subset of chronic *evolutions* identified inside SEDUITE.

When working on *sources* of information, we identified four chronic evolutions: *(i)* `cache` to implement cache mechanism when a source can slow down a process (*e.g.* external partner, temporary server crash) *(ii)* `shuffle` to shuffle a set of explicitly unordered information (*e.g.* a set of scrapped pictures), *(iii)* `toInfo` to transform an arbitrary data into one of type `Information` using an XSL meta–transformation and *(iv)* `truncate` to restrict the cardinality of an information set to a given size. When defining (or enhancing an existing one) a *provider*, we identify three evolutions: *(i)* `addSource` to add a new source of information, *(ii)* `dry` to dry up a source when time is over and *(iii)* `multiCalls` to allow a source to be called several times (*e.g.* a device broadcasting weather data for several cities instead of a single one).

[1] `http://www.lifl.fr/faros`
[2] `http://www.jseduite.org`

Performing these *evolutions* inside the system relies on the architect knowledge of those patterns. Even if those chronic *evolutions* are well documented, it is unrealistic to believe that everybody will follows the guidelines (SEDUITE is developed by ten different stake-holders, on three different continents). It rapidly results into an unmaintainable system where in front of **identical** situations, each architects leaves its own footprint on the architecture to propose an **equivalent** solution.

3 Composition of Orchestration Fragments

ADORE (*Activity moDel supOrting oRchestration Evolution*) is a platform designed to support orchestration evolutions. A dedicated meta–model supports the definition of orchestrations and orchestration fragments at a higher abstract level than usual formalism (*e.g.* BPEL). The representation used by ADORE supports the definition of a composition algorithm able to weave fragments into base orchestrations.

Orchestrations defined as BPEL code (based on the ADORE language restriction explained in next section) can be transformed into their representations conforming to the ADORE meta–model. The reciprocal transformation generates BPEL entities from an ADORE representation (using a technique similar to topological sort) to reach legacy servers infrastructure.

3.1 The ADORE Meta–Model

In ADORE, an orchestration of services is defined as a partially ordered set of activities. The different types of activities that can be defined in ADORE are a subset of the types of activities defined in the BPEL industrial specifications. These include *(i)* service invocation (denoted by `invoke`), *(ii)* variable assignment (`assign`), *(iii)* fault reporting (`throw`), *(iv)* message reception (`receive`), *(v)* response sending (`reply`), and *(vi)* the null activity, which is used for synchronization purpose (`nop`). In an ADORE process model, each process starts with a `receive` activity and ends with a `reply` activity.

An activity is identified by an unique identifier. Activities can use zero or more inputs and outputs. Unlike UML activity diagrams, in which an activity can have a nested structure, an ADORE activity is always primitive. A more complete description of the ADORE meta–model can be found on the project web site[3]. Figure 1 represents this meta–model as an UML class diagram.

We also define a graphical syntax to represent ADORE orchestrations. A box represents an activity inside the orchestration, and an arrow between two boxes means that the targeted activity of the arrow is allowed to start at the end of the source one. A label on an arrow represent a condition. Figure 2 represent a simple provider (which aggregates internal news and restaurant menu) using both formalisms (graphical and tuples).

[3] `http://www.adore-design.org`

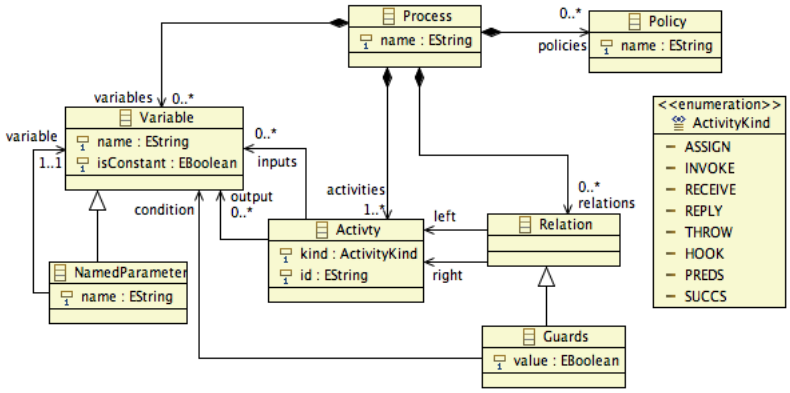
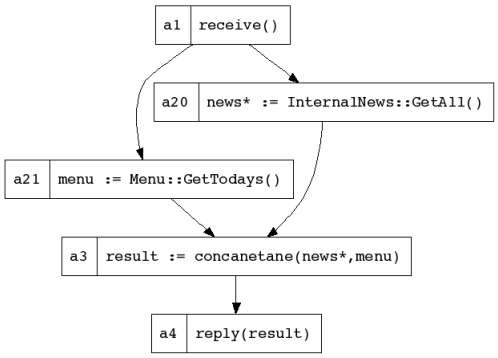


Fig. 1: ADORE meta–model (formal & simplified class–diagram)



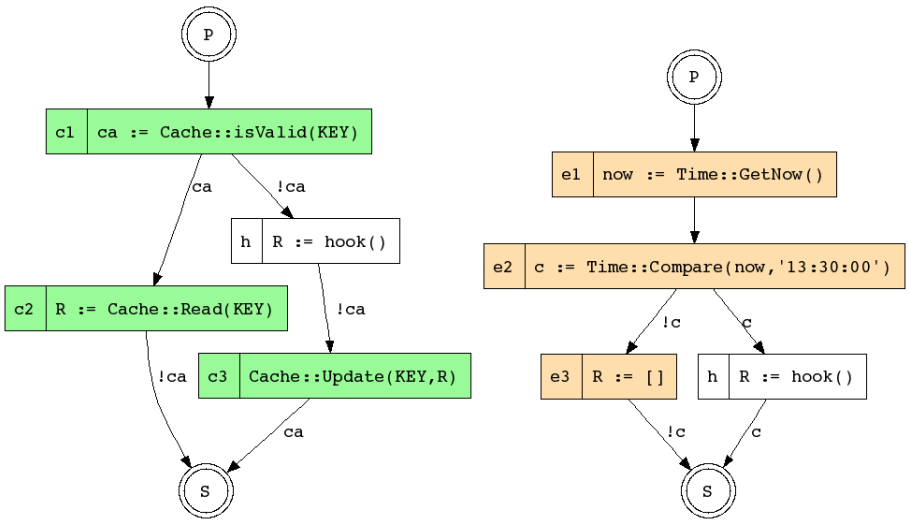
(a) ADORE graphical notation for p

$$
\begin{aligned}
p &\equiv (\{a_1, a_{20}, a_{21}, a_3, a_4\}, \\
&\quad \{a_1 \prec a_{20}, a_1 \prec a_{21}, \ldots, a_3 \prec a_4\}, \emptyset) \\
a_1 &\equiv (a_1, receive, \emptyset, \emptyset) \\
a_{20} &\equiv (a_{20}, invoke(internalNews, getAll), \\
&\quad \emptyset, news) \\
a_{21} &\equiv (a_{21}, invoke(menu, getTodays), \\
&\quad \emptyset, menu) \\
a_3 &\equiv (a_3, assign(concatenate), \\
&\quad \{news, menu\}, \{result\}) \\
a_4 &\equiv (a_4, reply, \{result\}, \emptyset)
\end{aligned}
$$

Fig. 2: ADORE example: p, a SEDUITE provider

3.2 Expressing Business–Driven Fragments Using ADORE

Defining an evolution in this context means to define a process *fragment* which enhances the behavior of an existing process activity. The fragment can interact with the (eventually unknown) targeted activity t_a in several ways, using three special activities: *(i)* the `hook` activity reifies t_a in the evolution context (providing an access to input and output variables), *(ii)* an activity $\mathbb{P}$ representing all immediate predecessors of t_a and *(iii)* an activity $\mathbb{S}$ representing all immediate successors of t_a. Fig. 3 represents two evolutions using our graphical syntax: *(i)* the `AddCache` evolution used inside SEDUITE to avoid slow response time and *(ii)* the `Dry` evolution used to dry up a source after a given hour (*e.g.* lunch menu is useless after lunch time).



(a) E_c:`AddCache` best-practice (b) E_d: `Dry` best-practice

Fig. 3: Expressing best-practices as ADORE evolutions

To integrate an evolution into an existing process we *unify* `hook`, $\mathbb{P}$ and $\mathbb{S}$ activities with t_a and its related activities. The unification is defined as a set of substitution σ [8] which perform modifications on the legacy process. A conflict detection algorithm can detect incoherent structure inside the builded process (*e.g.* concurrent write accesses to a shared variable) and ask for complementary knowledge to solve conflict. The resulting process is then transformed into a concrete BPEL process and then deployed inside an application server.

3.3 Evolution Composition Algorithm

When an architect wants to apply a *set* of evolutions $E \equiv \{e_1, \ldots e_n\}$ on a given targeted activities (*e.g.* an information source should be cached, truncated and shuffled), evolutions must be composed before being integrated into the targeted process. As a full description of this algorihtm is out of the scope of this paper (see [9]), we propose an informal and textual description: to perform the composition of behavioral evolutions, the algorithm starts to unify all `hook`, $\mathbb{P}$ and $\mathbb{S}$ activities inside E elements. Then, it propagates guards relations (conditional executions of activities) on the builded evolution. It results in a single and new evolution $e \equiv Merge(E)$. Fig. 4 illustrates the merge of the two previously described evolutions. The conflict detection mechanisms is executed over this process to detect incoherent structures. The resulting process can be integrated into an existing orchestration using the previously described technique.

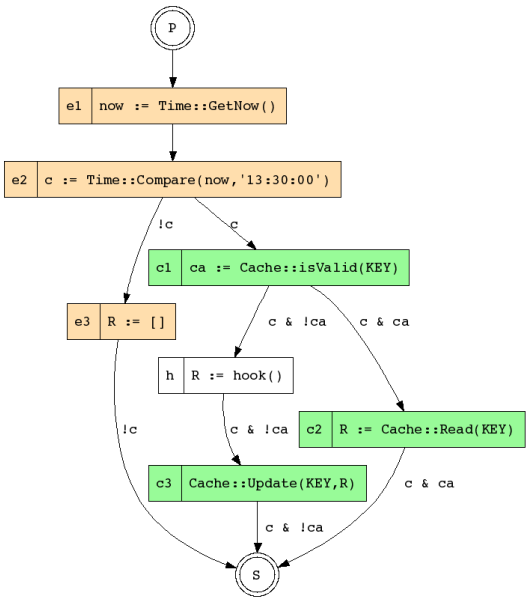


Fig. 4: $E' \equiv Merge(\{E_d, E_c\})$

4 Validation & Implementation

This section presents some experiments using the approach and then describes the implementation of the platform. As SEDUITE is a new platform with a

medium complexity size, we plan to perform further validations on a more complex system. Grid computing community is a good field of experience as it intensively relies on domain–dedicated data–flow to express intensive computations.

SEDUITE Case Study Results: We use this approach to model the two complete SEDUITE systems. In this paper, we focus on POLYTECH'SOPHIA system[4]. POLYTECH'SOPHIA SEDUITE software is composed by 15 elementary Web Services implemented in JAVA (representing more than $8KLoC$[5]), six BPEL orchestrations (almost $2KLoC$ and $14KLoC$ of WSDL and XSD artefacts) and two providers (more than $2KLoC$ and $11KLoC$ of artefacts).

Table 1 shows the usage of the identified *source* evolutions into SEDUITE orchestration. For each orchestration, it indicates which evolution were applied, and the cardinality of the activity set $\mathcal{A}^\star$ before (*resp.* after) the merge process. The Δ_f column represents the growth factor of the activity set. The average Δ_f is $1,61$, with a maximum of 2 for the `FeedReader` orchestration. This result shows that up to 50% of a process can be defined as the composition of evolution in this context.

–	Evolutions				$\lvert\mathcal{A}^\star\rvert$		
Orchestration	`AddCache`	`Shuffle`	`toInfo`	`truncate.`	before	after	Δ_f
`BusLocalizer`			×		5	6	1.2
`RssFeedReader`	×		×		4	8	2
`ImageScraper`		×	×	×	5	8	1.6
`RestaurantMenu`	×		×		6	10	1.7
`PictureAlbums`	×	×	×	×	9	15	1.7
`TvShows`			×		4	5	1.3
`Weather`	×		×		5	9	1.8

Table 1: Evolution usage in SEDUITE orchestrations

Table 2 shows the usage of evolutions to define a given provider: `SchoolProvider`. This provider is the more complex defined inside SEDUITE as the information set retrieved by this provider is broadcasted to a public screen in the main hall (so it handles an important set of sources). It aggregates ten different sources, each of them was added using a specialization of the `AddSource` evolution. For ten sources, only three do not use evolutions to enrich their information handling logic[6]. The seven others use one or two evolutions to customize the information

[4] As it is more complex than the other one.

[5] $1KLoC \equiv 1000$ lines of relevant code.

[6] These sources (`InternalNews`, `StudentConvocation` & `Weather`) broadcast dedicated information. As a consequence, they are defined as internal and dedicated services, so they do not need any others adaptation

retrieval process, which show the coverage of the method. The final provider is composed of 60 activities, where 58 ($\approx$ 96%) are automatically added and ordered.

Source	AddCache	DryUp	MultiCalls	Shuffle
Absences	×			
BreakingNews				×
BusLocalizer			×	
Calendar	×		×	
ImageScraper	×		×	
InternalNews				
Menu		×		
StudentConvocation				
TvShows	×	×		
Weather				

Table 2: `SchoolProvider` & Evolutions

Implementation SEDUITE reference implementation[7] uses JAX-WS Web Services as elementary bricks and BPEL 2.0 orchestrations to define complex business processes. The ADORE meta–model is implemented following the Model Driven Engineering (MDE) paradigm, using ECLIPSE EMF [10] as technological layer. A bridge between ADORE and an industrial BPEL meta–model (WTP project) is defined through a model transformation written using the KERMETA language [11]. As merge algorithms are defined in an inductive way and rely on logical properties, we use the PROLOG language to implement and validate them. Porting these algorithms into an object oriented language as JAVA or KERMETA (to bundle them in an evolution dedicated graphical modeling tool) is an ongoing work.

5 Related Work

Douence defines Aspect Oriented Programming (AOP) in [12] as *"a set of language mechanisms which enable the introduction of non anticipated functionalities in a base application"*. The AO4BPEL engine [13] brings aspect–oriented functionalities to an existing orchestration server. An aspect is supposed to be automatically woven on a given code without any additional information. When multiples aspects must be woven at the same place (so called shared join points [14]) existing solutions work at the code level (*e.g.* adding an order between two

[7] `http://www.jseduite.org`

aspects). Interactions between aspects (*i.e.* conflicts in ADORE terms) should then be managed at the level of the aspect definition, directly inside the advice code. Development by evolution composition is different, as we focus on the composition problem when a set of evolution must be applied at the same point. The composition algorithm is based on composition of a set of evolutions and doesn't depend of any order. Conflict detection steps ensure that the evolutions can be composed and that the build orchestration is valid.

Our work can also be compared with AOP works on mixins and traits [15]. However when these approaches rely on a composition based on multiple inheritance and code injection, evolution composition is based on the composition of graphs of activities and can generate new activities and relations.

Previously described approaches are *code–driven*. To help business experts and let them define business processes, higher level mechanisms and formalism were described and normalized. The business process community made several attempts to handle evolution and variability of these processes, like generic workflows [16] or process families [17]. These techniques put the focus on the representation of such concerns, without addressing the composition of business processes problematic. This issued is solved in ADORE through the composition algorithm.

6 Conclusions

In this article, we explain how an evolution framework called ADORE can be used to capitalize architects knowledge. This knowledge is reified as *evolutions* and then composed to build new architectures. The methodology is illustrated on a medium-complexity existing software called SEDUITE. This software is a validation platform of the FAROS research project (French national consortium) which aims to build reliable SOA through the composition of evolutions. Further validation of the methodology will be done on grid–computing data–flow for medical imaging.

Composition algorithm always focus on *enriching* an existing orchestration. We never address the evolution retract problem in an incremental approach. As far as we are in ADORE implementation, retracting an evolution means re–computing the global result from scratch instead of reasoning on the *delta* introduced by this evolution retract. An immediate perspective is to enrich evolution algorithms to take care of these kinds of considerations.

For now, the proposed methodology is designed for software architects. But when a business domain is restricted to its pure essence (CIM following MDE vocabulary), it should be manipulated by users who are not computer scientists but business experts. A long term perspective is to capture the information broadcasting domain into a dedicated meta–model and then write model transformation that reaches the proposed methodology at the PIM level. This abstraction will allow SEDUITE end user (*e.g.* schools' headmaster) to build their own dedicated providers without knowing anything about the underlying system: they will just express their sources set and handling policies at CIM level, and the

composition algorithm will do the composition at PIM level before generating *ready–to–deploy* BPEL orchestrations.

Acknowledgments This project is partially funded by the French Research Agency (ANR) through the FAROS project. The ADORE framework is one of the platforms targeted by FAROS. The SEDUITE software (through its reference implementation) is used as one of the validation platform.

References

1. Peltz, C.: Web services orchestration and choreography. Computer **36**(10) (2003)
2. MacKenzie, M., Laskey, K., McCabe, F., Brown, P.r., Metz, R.: Reference Model for Service Oriented Architecture 1.0. Technical report, OASIS (February 2006)
3. W3C: Web service glossary. Technical report, W3C (2004)
4. Arsanjani, A., et al: SOMA: a Method for Developing Service-Oriented Solutions. IBM Syst. J. **47**(3) (2008) 377–396
5. White, S.A.: Business Process Modeling Notation (BPMN). IBM Corp. (2006)
6. OASIS: WS Business Process Execution Language v2.0. Technical report (2007)
7. Papazoglou, M.P., Heuvel, W.J.V.D.: Service Oriented Design and Development Methodology. Int. J. Web Eng. Technol. **2**(4) (2006) 412–442
8. Stickel, M.E.: A Unification Algorithm for Associative-Commutative Functions. J. ACM **28**(3) (1981) 423–434
9. Mosser, S., Blay-Fornarino, M., Riveill, M.: Web Services Orchestration Evolution : a Merge Process for Behavioral Evolution. In: 2nd European Conference on Software Architecture (ECSA'08), Springer LNCS (2008)
10. Merks, E., Eliersick, R., Grose, T., Budinsky, F., Steinberg, D.: The Eclipse Modeling Framework. Addison Wesley (2003)
11. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented meta-languages. In: Proc. of MODELS/UML'2005. LNCS, Jamaica, Springer (2005)
12. Douence, R.: A restricted definition of AOP. In Gybels, K., Hanenberg, S., Herrmann, S., Wloka, J., eds.: European Interactive Workshop on Aspects in Software (EIWAS). (September 2004)
13. Charfi, A., Mezini, M.: Aspect-oriented web service composition with ao4bpel. In: ECOWS. Volume 3250 of LNCS., Springer (2004) 168–182
14. Nagy, I., Bergmans, L.M.J., sit, M.A.: Composing aspects at shared join points. In Hirschfeld, R., Kowalczyk, R., Polze, A., Weske, M., eds.: Proceedings of International Conference NetObjectDays, NODe2005, Erfurt, Germany. Volume P-69 of Lecture Notes in Informatics., Bonn, Germany, Gesellschaft fuer Informatik (GI) (September 2005) 19–38
15. Scharli, N., Ducasse, S., Nierstrasz, O., Black, A.P.: Traits: Composable Units of Behavior. In: ECOOP 2003 – Object-Oriented Programming. Volume 2743/2003 of Lecture Notes in Computer Science., OGI School of Science & Engineering , Oregon Health and Science University, Springer (November 2003) 327–339
16. van der Aalst, W.M.P.: Generic workflow models: How to handle dynamic change and capture management information? In: CoopIS, IEEE Computer Society (1999)
17. Schnieders, A., Puhlmann, F.: Variability mechanisms in e-business process families. In Witold Abramowicz, H.C.M., ed.: 9th International Conference on Business Information Systems (BIS 2006), May 31 - June 2, 2006, Klagenfurt, Austria, Springer-Verlag (2006) 583–601