Towards correct evolution of components using VPA-based aspects

Dong Ha Nguyen and Mario Südholt

OBASCO project; Ecole des Mines de Nantes - INRIA, LINA; Nantes, France
{Ha.Nguyen, Mario.Sudholt}@emn.fr

Abstract. Interaction protocols are a popular means to construct correct component-based systems. Aspects that modify such protocols are interesting in this context because they support the evolution of such components. A major question then is whether aspect-based evolutions preserve fundamental notions of correctness, in particular compatibility and substitutability, of components. In this paper we discuss how such component correctness properties can be proven in the presence of aspect languages of limited expressiveness. Concretely, we show how common evolutions involving VPA-based aspects [12] can be proven correct directly in terms of operators of the aspect language. Furthermore, we present first ideas of how to use existing model checkers for the automatic verification of such properties.

1 Motivation

Interaction protocols are a popular means to construct correct component-based systems and document them [19, 6]. Relying on explicit protocols, evolution of component-based systems can be frequently expressed in a concise manner using aspects that modify such protocols [12].

A major question for the evolution of component-based systems is whether evolution preserves compositional properties of these systems, in particular compatibility and substitutability of components, two fundamental notions that are typically defined in terms of subset relationships of trace and failure sets admitted by the original and evolved versions of a system [19, 13]. Currently, almost all component-based systems with interaction protocols have used finite-state protocols; only few work has explored the preservation of compositional properties in the context of aspects modifying such interaction protocols.

In this paper we consider how compositional properties can be defined and verified in the context of the evolution of components that are equipped with a more expressive brand of interaction protocols, protocols defined in terms of Visibly Pushdown Automata (VPA) [2]. VPA allow to define protocols that include well-formed nested contexts, such as correct nesting of recursive calls to and returns from a server. VPAs are strictly more expressive than finite-state automata

* This work has been supported by AOSD-Europe, the European Network of Excellence in AOSD (http://www.aosd-europe.net).

(which generate regular languages) but strictly less so than pushdown automata (which generate context-free languages). In contrast to finite-state based systems, VPA-based protocols allow (some) nested terms to be correctly matched without having to restrict the nesting depth. In contrast to pushdown automata, VP languages are closed under all basic operations, including intersection and complement, and all basic decision problems are decidable. As the main contribution of this paper, we discuss how compatibility and substitutability properties of component-based applications can be proven if interaction protocols are subject to evolution using VPA-based aspects [12].

Concretely, we motivate and sketch three extensions to the VPA-based aspect language that are useful for the evolution of component systems: a more general definition of sequence pointcuts, a new pointcut operator that allows nested contexts to be matched if their depths exceed a threshold and a new advice construct that allows to close an open nested context. Furthermore, we introduce several proof methods that can be used to prove the preservation of compositional properties if the resulting aspect language is used for component evolution. Finally, we present preliminary results how to use existing model checking techniques for the automatic verification of such properties.

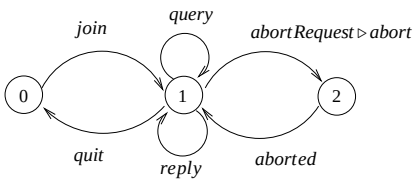
The paper is structured as follows. In Section 2, we motivate and sketch three extensions to our VPA-based aspect language. We introduce corresponding proof methods for software components in Sec. 3. VPA-based aspects and model checking techniques are the subject of Sec. 4. Finally, we give a conclusion and present future work in Sec. 6.

2 VPA-based aspects for component evolution

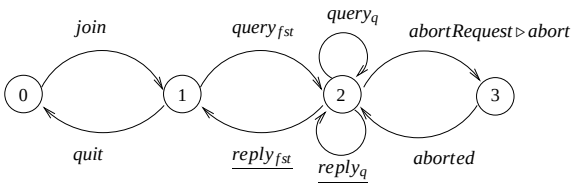
In the following we illustrate the use of expressive, *i.e.,* non-regular aspect languages in the context of (distributed and recursive) P2P algorithms. Fig. 1(a) shows a protocol which allows nodes to join or quit a P2P network, repeatedly execute recursive queries and abort queries if requested (by means of an advice $abortRequest \triangleright abort$).

The regular aspect on the left hand side does not enforce an important restriction: abort requests should only be allowed if there is at least one on-going query (the number of replies occurring at state 1 may equal or even exceed the number of queries in the regular case). The VPA-based aspect shown on the right hand side, however, ensures this property by distinguishing the first query and the matching reply events from the remaining ones by associating stack symbols to transitions (in the figure, stack symbols are set as indexes to execution events and matching replies are underlined). Abort requests therefore may occur only in contexts where at least one query is open. Furthermore, VPAs ensure that there cannot be more *abortRequest* events than query events.

In previous work [12], we have proposed a language for VPA-based aspects and a corresponding execution library. This language allows to define pointcuts in terms of paths in a VPA whose matching during execution of a base application, *e.g.,* an OO program, triggers the execution of advice as illustrated in 1(b). We



(a) Aborting on-going queries (regular)



(b) Aborting on-going queries (VPA)

now illustrate that VPA-based aspects are useful for the evolution of component-based systems by presenting evolutions that can be supported using three new operators that extend our original aspect language.

Evolution of the recursive structure of P2P algorithms using pointcut operators. Evolution of distributed algorithms, such as P2P algorithms, often aims at the optimization of the underlying traversal strategy. A simple example of a corresponding heuristic is to perform a more superficial but faster search on nodes whose distance from the root node exceeds a certain threshold. Since VPAs faithfully allow to define the depth of nested terms, such heuristics can be directly expressed using a pointcut operator $D_m^{>k}$ that matches only calls to m that occur at a depth larger than k. For example, the following aspect caches queries at depth greater than 5 (where $\mu a. \ldots ; a$ denotes recursion in VPA-based aspects):

$$\mu a.\ D^{>5}_{query_q} \rhd getCacheValue\ ;\ a$$

Accomodating new execution events through general sequencing of pointcuts. The evolution of component systems frequently requires to cope with new execution events, either by abstracting from them (*i.e.,* allow interleaving of such events on the protocol level) or, to the contrary, forbid the occurrence of such events. Current aspect languages for protocols typically include a sequence operator ; on the pointcut level, such that terms $a; b$ allow either no interleaving (*i.e.,* the term corresponds to a single-step transition as, *e.g.,* JAsCo's stateful aspects [17]) or interleaving of arbitrary events between occurrences of a and b (as the stateful aspects of [4, 12]).

Using the latter semantics, the aspect

$$\mu a.\ join\ ;\ query_q \rhd createSession\ ;\ a$$

(repeatedly) creates sessions once the current node has joined a P2P network, occurrences of arbitrary events followed by a query. However, the occurrence

of events, *e.g.,* accessing the result of a query before execution of the query, cannot be ruled out with this form of sequencing alone. Relying only on the first semantics, individual transitions yield awkward formulations of non-trivial protocol-modifying aspects because all interleaving of events has to be defined explicitly.

In order to allow the concise definition of protocol evolution by arbitrary interleaving as well as through the (mandatory) absence of interleaving we have introduced a general sequencing operator $;_{\mathcal{I}}$ where $\mathcal{I}$ specifies the set of events, possibly $\emptyset$, that may be interleaved between the argument events.

The evolution consisting in forbidding previous accesses to the query result can then simply be expressed as:

$$\mu a.\ join\ ;_{\neg accessResult}\ query_q \rhd createSession\ ;\ a$$

Handling of error conditions using advice operators. The evolution of component-based systems frequently consists in the introduction of behavior to correctly handle error situations. In the case of recursive distributed algorithms error handling may involve the introduction of events that close a number of open recursive calls in order to skip the traversal of part of the underlying distributed network in which an error occured. Using VPA-based aspects such error handling strategies can be expressed using the advice-level operation $closeOpenCall_m$ that closes the open call to m: pointcuts matching on nested contexts can then be used to restrict the application of such advice to appropriate parts of the network. The following example illustrates the use of a closing operator to add a number of "fake" replies to queries when the query exceeds a given connection timeout (where $\Box$ denotes the choice between alternatives):

$$\mu a.\ query_f; (\underline{reply_f}\ \Box\ (connectionTimeOut \rhd closeOpenCall_{query_f}))\ ;\ a$$

3 Preservation of compositional properties

In this section we address the problem whether compositional systems that are subjected to evolution by VPA-based aspects can be proven to preserve fundamental composition properties. Our main point is that, in contrast to general aspect languages such as AspectJ, VPA-based aspect programs are amenable to formal correctness proofs.

Figure 1 illustrates the underlying model of component evolution and the compositional properties we consider. Starting from two protocols p_1, p_2 that constrain the interactions of two collaborating components C_1, C_2 a VPA-based aspect $\mathcal{A}$ is applied to p_2 yielding the protocol p_3 that defines the interactions of the component C_3 after evolution. As indicated in the figure we are interested in two fundamental correctness properties for components, compatibility and substitutability (see, *e.g.,* [13]).

Generally, *e.g.,* if turing-complete pointcut and advice languages are used for component evolution (as in AspectJ where arbitrary Java methods may be called in `if`-pointcuts and advice), such component properties cannot be proven

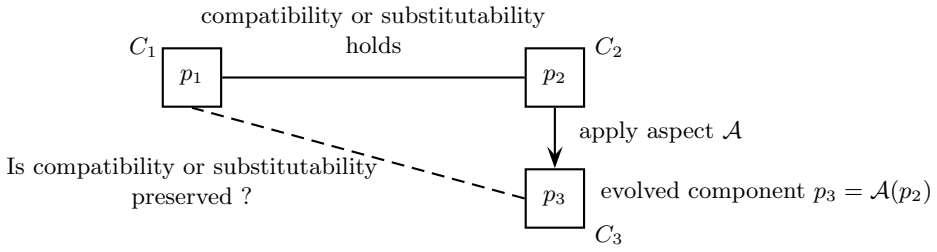


Fig. 1. Checking for preservation of compatibility/substitutability

formally. Furthermore, even in specific cases where a proof is possible, it can typically be performed only in terms of the woven program and not simply in terms of the aspects themselves. VPA-based aspects, however, support formal proofs of such properties because of their limited expressiveness and allow some important properties be proven simply by considering properties of the aspect language only. To this end we propose to exploit the "domain specific" characteristics of VPA-based aspects: proofs over nested contexts as well as regular structures can be performed directly in terms of corresponding features of our pointcut (indexed calls) and advice language (*closeOpenCall*).

Concretely, we demonstrate in the following three different types of proofs of property preservation that are supported by VPA-based aspects:

P1) Proofs that depend only on the properties of the aspect language, *i.e.,* that can be performed in terms of the evolution aspect $\mathcal{A}$ only.
P2) Proofs that can be performed in terms of $\mathcal{A}$ and properties of *classes* of protocols to which p_1 and p_2 belong.
P3) Proofs that require full knowledge of $\mathcal{A}$ and p_1–p_3.

We use standard trace-based notions of compatibility and substitutability [19] in this paper. Two protocols are compatible if they do not give rise to any conflict during execution, *i.e.,* no unexpected message is received during collaboration of two components according to their respective protocols. Substitutability of components is defined using trace set inclusion: protocol p_1 is substitutable for p_2 if its trace set is a superset of the trace set generated by protocol p_2. Since VPAs are closed under complement ("negation") it is, however, possible to apply the proof methods to more expressive notions of composition properties, for instance substitutability in the presence of failures [13].

In the following we will present three examples that illustrate the different proof types introduced above.

P1: supporting evolution of error handling. VPA-based aspects are unique (in particular compared to finite-state based approaches) in being able to handle a large class of traversals of distributed recursive algorithms, such as P2P algorithms. Frequently, error handling in such algorithms consists in terminating the exploration of some part of the network and search elsewhere. The action *closeOpenCall*(m) that we have introduced in the advice language directly supports such error strategies by allowing to close a nested call of the method m.

We can exploit the precisely defined semantics and limited effect of the action *closeOpenCall* to prove some corresponding properties simply in terms of its definition. For example:

> If p_1,p_2 are protocols that recurse using m, p_2 is substitutable for p_1 and aspect $\mathcal{A}$ employs *closeOpenCall* to add returns of m at the end of the execution of protocol p_2, then the adapted protocol p_3 is substitutable for p_1.

P2: proving compatibility for depth-cutting heuristics. Recursive distributed algorithms frequently do not unconditionally stop traversals at the top level, but typically do so only in specific contexts. A common example are heuristics that are formulated in terms of the traversal depth from the node where the search has been initiated. Since VPA-based aspect allow the explicit definition of aspects in terms of the nesting depth using the pointcut operator $D^{>k}_{m_c}$, corresponding compositional properties can be proven in terms of properties of this operator and classes of protocols to which it is applied. For example:

> If
> - p_1 belongs to the class of recursive protocols that repeatedly allows recursive remote calls and returns in m: $\mu a.m_c \,\Box\, \underline{m_c}\ ;\ a$,
> - p_2 belongs to the class of protocols that include a remote call to m,
> - p_1, p_2 are protocol compatible and
> - aspect $\mathcal{A}$ employs a depth-defining operator $D^{>k}_{m_c}$ applying over p_2
>
> Then p_1 and $\mathcal{A}(p_2)$ are also compatible.

This property holds because the aspect may only cut calls to m from traces of p_2: the resulting traces remain compatible with those admitted by p_1.

P3: proving substitutability in terms of p_1,p_2 and $\mathcal{A}$. Let us reconsider protocols p_1,p_2 as in the first example *i.e.,* p_2 represents a less restrictive recursive protocol than p_1 and p_2 is substitutable for p_1. Assume that now we would like to adapt protocol p_2 in order to cut the depth of queries to k using an aspect with a depth-defining operator $D^{>k}_{m_c}$. In this case the resulting protocol p_3 is in general not substitutable for p_1, since p_1 may admit calls of depth deeper than k. By an analysis of p_1, we may find that the depth limit of p_1 is q and $q \leq k$: we can then prove that p_3 is actually substitutable for p_1.

4 Towards model checking of VPA-based properties

We now consider the problem of using model checking techniques as a support for proving the preservation of properties of component systems that are subject to aspect-based evolution. We present first preliminary results on two main issues concerning the application of model checking: (i) adaption of the default model checking procedure to the verification of systems with VPA-based aspects and (ii) selection of an appropriate model checker.

Goldman and Katz [9] have introduced a formal framework for verifying the correctness of an aspect in a modular way. The general ideas of that approach are as follows. Assume that an aspect is defined by its pointcut designator ρ and advice A represented by a single state machine. Furthermore, the assumptions of the aspect about the base programs into which it may be woven are explicitly specified in form of an LTL formula ψ from which a tableau T_ψ can be constructed. Weaving two state machines respectively represented by the tableau T_ψ and aspect advice A according to pointcut ρ results in an augmented state machine for the composed system $\widetilde{T_\psi}$. Then a model checker is employed to verify whether the system $\widetilde{T_\psi}$ satisfies a property φ over the complete system. This means that if we could establish that a specific base program satisfies the assumptions ψ we do not have to run the verification process again in case it has already been done for the combination A, ρ, ψ, and φ before. Moreover, the real composed system has never to be model checked and thus this verification approach achieves modularity.

We plan to adapt the above framework for model checking to the correctness of VPA-based aspects. The underlying idea of the adaptation is to define an abstraction of VPA-based programs and aspects into regular systems and then apply model checking techniques. Application of existing model checkers requires to fix the depth of recursive contexts matched through VPA-based expressions, which implies to sacrifice some accuracy by using a suitable approximation. Through an extension of the approach of Goldman and Katz we intend to provide a means to efficiently model check abstractions of recursive VPA-based structures in which the depth is fixed but arbitrary.

Table 1. Some model checkers and corresponding properties

Model checker	System model	Property	Remarks
SPIN	Promela (SPIN's language)	LTL	popular
NuSMV	Finite state machine	CTL,LTL	popular
CBMC	C source code		
Bandera	Java source code		no longer developed
UPPAAL	Real-time automata	CTL	
Verus	Real-time automata	CTL	no longer developed

A second important factor for such an approach is the specific model checker being selected because of their rather different features, such as their system models, that are more or less suitable for our endeavor. Table 1 presents a list of some well-known model checkers and some of their features.

Three properties of model checkers we are particularly interested in are (i) how the input system can be modeled, (ii) what types of properties are supported, and (iii) how large of a system on which it can model check.

System model. All model checkers on Table 1 input system models as textual descriptions of state machines in their respective, specific formats. As previously

mentioned, since we are dealing with VPA-based systems, we have to transform them to less expressive automata then encode them in the input formats which are supported by these model checkers. This can be automatically done by a transformation tool, but should be easier for checkers using input languages similar to state machines (such as NuSMV and SPIN) than for checkers using general purpose languages (CBMC and Bandera).

Property specification. Most of the model checkers support property specified in some variants of LTL or CTL logics. Among them, model checker NuSMV provides the most flexibility by accepting both LTL and CTL properties. Checkers that are geared towards the handling of specific classes of properties (e.g., real-time properties in the case of UPPAAL and Verus) seem less appropriate for our approach.

Scalability. Since the abstraction from VPA-based properties to regular systems generates large finite-state machines (that are of a very specific form), the scalability of model checkers is an important criterion for the feasibility of our approach. However, while the first two features, system and property specification of model checkers, can be evaluated simply, it is more difficult to have a comparative view on the scalability feature since verification tools are typically designed and optimized for different specific domains. Among current model checkers, SPIN and NuSMV are highly recognized for their effectiveness in the presence of large systems.

Taking all the factors into consideration, we have chosen to perform first experiments on the model checking of VPA-based evolution properties using SPIN and NuSMV.

5 Related work

There is few related work on aspect-based evolution for component-based systems that considers the preservation of correctness properties for those systems after being changed by aspects. As to the best of our knowledge, this proposal is the first exploiting formal methods to investigate the preservation of compositional properties such as compatibility and substitutability for component-based systems that are subject to evolution by protocol-modifying aspects. However, our work still shares common interests with a large body of work covering aspects, components, and applications of formal methods on analysis and verification.

There are some approaches which consider aspect languages that support protocols, most notably [1, 5, 18]. Approaches [1, 5] feature regular aspect languages and a framework for static analysis of interaction properties. The language introduced by Walker and Viggers[18], one of the very few approaches providing non-regular (but not turing-complete) pointcut languages, proposes tracecuts which provide a context-free pointcut language. However, all of the above approaches do not use the language for an integration of aspects and components

or explore the problem of property-preserving for systems that have protocols being modified by aspects. Farías [7] has proposed a regular aspect language for components that admits advice modifying the static structure of protocols and considered proof techniques for the resulting finite-state based aspects.

There exist a large number of proposals that aim at applying AOP over component-based systems, *e.g.,* [8, 16, 14]. However, the aspect languages in those approaches do not provide explicit support for component protocols. Some of these approaches consider component compatibility, however, in a limited sense to our work: aspects are usually employed in such work to transparently introduce adaptation to components and thus preserve component compatibility. Our approach, in contrast, focuses on preserving protocol compatibility even if aspects have visible effects on interaction protocols.

Few work on evolution of component protocols seems relevant to our work. Braccialia et al. [3] present a formal methodology for automatically adapting components with mismatching interacting behaviors *i.e.,* conflicts at the protocol level. Protocols considered there are expressed by using a subset of $\mu-$calculus. They do not consider how component properties can be proved in terms of proof methods that exploit properties of modification operators. Ryan and Wolf [15] investigate how applications can accommodate protocol evolution. However, this approach concerns mainly syntactic changes on protocols.

Another category of related work is the application of formal methods to analyse aspect systems, such as [10, 11]. Our approach differs from those approaches in that we exploit the protocol-based specificities of our aspect language to prove composition properties of software components.

6 Conclusion and future work

In this paper we have investigated the preservation of compositional properties in the context of aspect-based evolutions on components. We have shown that aspect languages of limited expressiveness admit formal proofs of fundamental compositional properties directly in terms of the aspect languages. Concretely, we have shown that VPA-based aspects support formal proofs of component compatibility and substitutability in the presence of aspect-based evolutions of recursive distributed algorithms. As a second contribution, we have introduced three extensions to our VPA-based aspect language that support common evolutions: a more flexible sequencing operator, a depth-defining pointcut operator and a closing operator for recursive calls. Finally, we have presented first ideas of how to use use existing model checkers for the automatic verification of such properties and evaluated the characteristics of some popular model checkers to this end.

In the future we intend to work on extensions of the aspect language for VPA-based properties, extend the set of component evolutions that can be handled with our approach and provide a working method for the automatic verification of such evolutions with existing model checkers.

References

1. Chris Allan, Pavel Avgustinov, Aske Simon Christensen, et al. Adding trace matching with free variables to AspectJ. In Richard P. Gabriel, editor, *ACM Conference on Object-Oriented Programming, Systems and Languages (OOPSLA)*. ACM Press, 2005.
2. Rajeev Alur and Parthasarathy Madhusudan. Visibly pushdown languages. In *Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing (STOC-04)*, pages 202–211, New York, June 13–15 2004. ACM Press.
3. Andrea Braccialia, Antonio Brogi, and Carlos Canal. A formal approach to component adaptation. *Journal of Systems and Software*, 2005.
4. R. Douence, P. Fradet, and M. Südholt. A framework for the detection and resolution of aspect interactions. In *Proc. of GPCE'02*, LNCS 2487, pages 173–188. Springer Verlag, October 2002.
5. Rémi Douence, Pascal Fradet, and Mario Südholt. Composition, reuse and interaction analysis of stateful aspects. In *Proc. of 3rd International Conference on Aspect-Oriented Software Development (AOSD'04)*, pages 141–150. ACM Press, March 2004.
6. Andrés Farías and Mario Südholt. On components with explicit protocols satisfying a notion of correctness by construction. In *International Symposium on Distributed Objects and Applications (DOA)*, volume 2519 of *LNCS*, pages 995–1006, 2002.
7. Andrés Farías and Mario Südholt. Integrating protocol aspects with software components to address dependability concerns. Technical Report 04/6/INFO, École des Mines de Nantes, November 2004.
8. Steffen Göbel, Chrstoph Pohl, Simone Röttger, and Steffen Zschaler. The COMQUAD component model — enabling dynamic selection of implementations by weaving non-functional aspects. In *Proceedings of AOSD'04*. ACM Press, 2004.
9. Max Goldman and Shmuel Katz. Maven: Modular aspect verification. In *TACAS*, pages 308–322, 2007.
10. Shmuel Katz and Marcelo Sihman. Aspect validation using model checking. In *Verification: Theory and Practice*, pages 373–394, 2003.
11. Shriram Krishnamurthi and Kathi Fisler. Foundations of incremental aspect model-checking. *ACM Trans. Softw. Eng. Methodol.*, 16(2):7, 2007.
12. Dong Ha Nguyen and Mario Südholt. VPA-based aspects: better support for AOP over protocols. In *4th IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*. IEEE Press, September 2006.
13. Oscar Nierstrasz. Regular types for active objects. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, chapter 4, pages 99–121. Prentice Hall, 1995.
14. Nicolas Pessemier, Lionel Seinturier, Thierry Coupaye, and Laurence Duchien. A model for developing component-based and aspect-oriented systems. In *Proceedings of the 5th International Symposium on Software Composition (SC06)*, volume 4089 of *Lecture Notes in Computer Science*, page 259273, Vienna, Austria, mar 2006. Springer-Verlag.
15. Nathan D. Ryan and Alexander L. Wolf. Using event-based translation to support dynamic protocol evolution. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 408–417, Washington, DC, USA, 2004. IEEE Computer Society.
16. Davy Suvée, Wim Vanderperren, and Viviane Jonckers. JasCo; an aspect-oriented approach tailored for component-based software development. In ACM Press,

editor, *Proc. of 2nd International Conference on Aspect-Oriented Software Development (AOSD'03)*, pages 21–29, March 2003.
17. W. Vanderperren, D. Suvee, M. A. Cibran, and B. De Fraine. Stateful aspects in JAsCo. In *Proc. of SC'05*, LNCS 3628. Springer Verlag, April 2005.
18. Robert J. Walker and Kevin Viggers. Implementing protocols via declarative event patterns. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-12)*, pages 159 – 169. ACM Press, 2004.
19. Daniel M. Yellin and Robert E. Strom. Protocol specifications and component adaptors. *ACM Transactions of Programming Languages and Systems*, 19(2):292–333, March 1997.