

# Property-preserving 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 [4] can be proven correct directly in terms of operators of the aspect language.

## 1 Motivation

Interaction protocols are a popular means to construct correct component-based systems and document them [7, 3]. Relying on explicit protocols, evolution of component-based systems can be frequently expressed in a concise manner using aspects that modify such protocols [4].

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 [7, 5]. 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) [1]. VPAs 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 (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 [4].

Technically, 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.

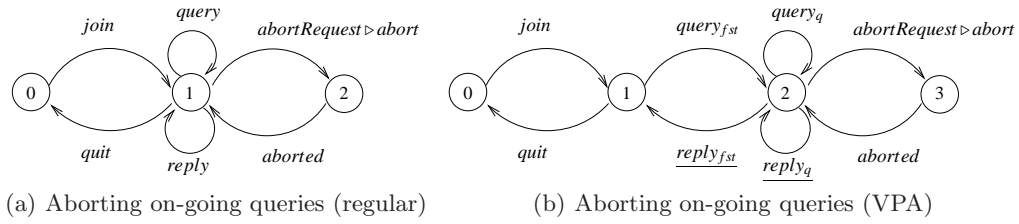
---

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

Finally, 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.

## 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 [4], 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 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  $\mu a. \dots ; a$  denotes recursion in VPA-based aspects):

$$\mu a. D_{query_q}^{>5} \triangleright getCacheValue ; a$$

*Accommodating 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 [6]) or interleaving of arbitrary events between occurrences of  $a$  and  $b$  (as the stateful aspects of [2, 4]).

Using the latter semantics, the aspect

$$\mu a. \text{join} ; \text{query}_q \triangleright \text{createSession} ; a$$

(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:

$$\mu a. \text{join} ; \neg_{\text{accessResult}} \text{query}_q \triangleright \text{createSession} ; a$$

*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 occurred. Using VPA-based aspects such error handling strategies can be expressed using the advice-level operation  $\text{closeOpenCall}_m$  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  $\square$  denotes the choice between alternatives):

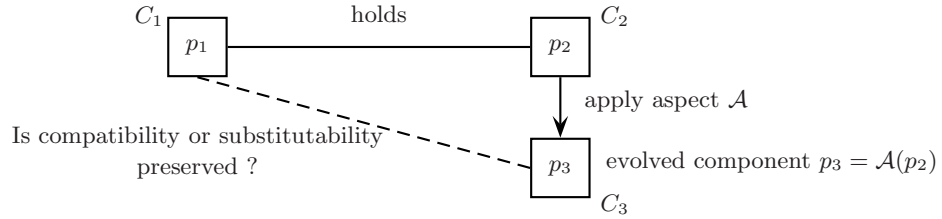
$$\mu a. \text{query}_f ; (\underline{\text{reply}}_f \square (\text{connectionTimeout} \triangleright \text{closeOpenCall}_{\text{query}_f})) ; a$$

### 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.*, [5]).

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 formally. Furthermore, even in specific



**Fig. 1.** Checking for preservation of compatibility/substitutability

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 [7] 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 [5].

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 aspects allow the explicit definition of aspects in terms of the nesting depth using the pointcut operator  $D_{m_c}^{>k}$ , 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  $m$ :  $\mu a.m_c \square \underline{m_c} ; a$ ,
  - $p_2$  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  $D_{m_c}^{>k}$  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  $D_{m_c}^{>k}$ . 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 Conclusion

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.

## References

1. 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.
2. 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.
3. Andrés Fariás 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.
4. 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.
5. 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.

6. 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.
7. 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.