A Closer Look at Aspect Interference and Cooperation

Cynthia Disenfeld Shmuel Katz

Department of Computer Science
Technion - Israel Institute of Technology
{cdisenfe,katz}@cs.technion.ac.il

Abstract

In this work we consider specification and compositional verification for interference detection when several aspects are woven together under joint-weaving semantics without recursion. In this semantics, whenever a joinpoint of an aspect is reached, the corresponding advice is begun even if the joinpoint is inside the advice of other aspects. This captures most of the possible aspect interference cases in AspectJ. Moreover, the given technique is used to capture cooperation among aspects, which enhances modularity. The extended specification and proof obligations should provide insight to the possible interactions among aspects in a reusable library.

Categories and Subject Descriptors D.2.1 [*Software Engineering*]: Requirements/Specifications; D.2.4 [*Software Engineering*]: Software/Program Verification—Correctness proofs, Model checking

General Terms Languages, Verification

Keywords Aspects, Joint-Weaving, Verification, Composition, Cooperation, Interference

1. Introduction

Aspects capture problems that may crosscut the application, such as logging, persistence, exception management, and others. Weaving several aspects within an application may lead to *interference*: given a set of aspects, each aspect on its own behaves as expected but when considering all aspects woven together the expected behavior is no longer achieved.

In this work, we extend existing results on interference detection, to treat the basic *joint-weaving* semantics seen in AspectJ. In joint-weaving semantics, whenever a joinpoint of an aspect is reached, the corresponding advice is begun even if the joinpoint is inside the advice of other aspects.

AOSD'12 March 25–30, 2012, Potsdam, Germany.


```
aspect Auth
before() : doTrans()
  Usr u = requestUsr()
  Pwd p = requestPwd()
  authenticated =  usrPwdExist(u,p)

aspect SaveCookie
after (Usr u, Pwd p) returning (boolean success):
    call(* usrPwdExist(..)) && args(u,p)
  if (success) saveCookie(u,p)

aspect EncryptPwd
Pwd around(): call(requestPwd())
  Pwd p = proceed()
  return encrypt(p)
```

Figure 1. Aspects `Auth`, `SaveCookie` and `EncryptPwd`

The program listing in Figure 1 shows AspectJ-style aspects to authenticate transactions on a website (`Auth`), to save a cookie on authentication success (`SaveCookie`), and to encrypt passwords (`EncryptPwd`), where the latter two are activated within the advice code of the first.

Note that `SaveCookie` does not affect the behavior of `Auth`. However `EncryptPwd` may cause an existing user and password not to be found anymore by `Auth` (because the password is now encrypted by the time the check is done). This interference is caused from the execution of `EncryptPwd` within `Auth`. In this paper, we will consider a precise representation of aspect specifications which allow using formal verification techniques to detect such interference.

If `EncryptPwd` assumes that every password satisfies some necessary constraints (such as a minimum number of characters, combination of letters, numbers and symbols), then another aspect may collaborate in order to guarantee that the assumptions of `EncryptPwd` hold. Such aspect collaboration will also be considered.

A restricted procedure for detecting interference among aspects using model checking is presented in MAVEN [8]. This method assumes that there are no joinpoints inside advice and is valid for *sequential weaving*, where the advice of an aspect is woven into the system at the joinpoints available so far, and then another aspect can be woven. (In addition, [8] considers simple cases in which joint weaving is equivalent to sequential weaving.)

We will analyze and present a compositional verification and interference detection technique for the case in which aspects may have new joinpoints within the advice of other aspects and that is valid for joint-weaving semantics. As explained in the following section, the verification technique is based on model checking state machines derived from AspectJ code. The method to be presented is for at most weakly-invasive aspects [12], that is they may return to a different state of the underlying system as long as the state was already reachable from some other execution. This assumption is in order to make notation simpler, but the same ideas can be applied for the general case, as explained at the begining of Section 5. We also assume that there is no recursion, i.e. an aspect is never executed under its own execution flow, not even indirectly.

The method predicates that aspects can be given a generalized assume-guarantee specification, where the underlying system and environment are assumed to satisfy the aspect's assumption, and the augmented system with the aspect should satisfy its guarantee. This form of specification allows building a library of verified aspects where possible interference has been analyzed in advance. Then, this library may be used in any system that satisfies the properties that the aspects assume, and the guarantees of the aspects hold without the need to perform any additional checks.

This work provides the following main contributions:

- understanding the problems that arise when considering joint-weaving semantics instead of sequential weaving. By giving examples it will be shown how existing mechanisms that rely on sequential weaving do not always detect interference when the joint-weaving model is used.
- treating joint weaving by distinguishing between *global* and *local* guarantees. When aspects may add joinpoints of other aspects, some aspects may not be aware of all the matching joinpoints in the state machine model of aspect weaving. Hence, global guarantees express properties that should hold even when not all aspects are aware of all joinpoints, and local guarantees express what is expected at those joinpoints of which the aspect is aware.
- treating aspects that contain joinpoints of other aspects by adding an *internal assumption* to the specification of every aspect A. The *internal assumption* of A represents what every other aspect B to be executed within the execution of A must satisfy.

  Some default internal assumption sorts are presented, e.g. when inserted aspects are assumed to satisfy an invariant; however they are not restrictive and any such assumption can be treated.

  Being aware of the need for internal assumptions when writing aspect specifications promotes a better understanding of the system, even when not applying formal verification techniques: aspect interactions, dependencies and cooperation requirements are conveyed by the internal assumptions.
- providing a compositional verification technique to verify non-interference in a set of aspects that may add new joinpoints of other aspects under joint weaving semantics. This technique takes into account the particular internal assumption, since certain defaults may lead to simpler checks.

  Given a library of aspects, they must be checked only once, and then for any system which satisfies the necessary aspect assumptions, the system augmented with the aspects is already proven to satisfy all aspect guarantees.

  Adding a new aspect to a library implies checking this aspect against all the other aspects in the library, but the proofs already done are still valid and used in the new proof of interference-freedom.
- extending the compositional verification technique for considering aspect cooperation.

  The correctness proof of collaborative aspects yields aspect dependencies, and also allows incrementally verifying as the system is being built - the assumptions of an aspect A are considered to hold, and then the necessary assistant aspects can be developed and verified to satisfy the assumptions of A.

The next section gives background on aspects, temporal logic, weaving semantics and the previous work on aspect verification that is extended here. Sections 3 and 4 describe the technique for detecting interference, including aspect specification and verification. The soundness proof is given in Section 5. Section 6 discusses cooperation. The technique applied to removed joinpoints is considered in Section 7. Section 8 presents related work and we conclude in Section 9.

2. Background

2.1 Aspects

Aspects allow describing crosscutting concerns of a system by defining in which states (*joinpoints*) a certain response should be woven, and what the response consists of (*advice*). *Pointcut descriptors* describe the joinpoints where the response should be woven. Each *advice* consists of the code to be executed when the pointcut is matched. In this work, we consider aspect advices given by state machines. Even though the examples are given in AspectJ, as mentioned before we assume the use of tools such as [5] that can transform code to state machines automatically. Creating such a finite-state model from Java code is a non-trivial task, and generally requires abstracting data domains and system states. Often, a careful abstraction can guarantee that if the specification holds for the abstracted model it also holds for the original version. Weaving an aspect A to a system S is then a state machine transformation, where for each state s in S

that represents a joinpoint of A the next states of s are now the entry points to the aspect execution. When A returns, it returns to a state representing the program point where it should go (if no exceptions were thrown) and the variable contents according to the changes that A has applied.

Aspects can be categorized according to the semantic transformation they make to the underlying system [12]:

Spectative aspects gather information, but do not change control flow or the values of the variables non-local to the aspect.

Regulative aspects may change the control flow, but do not change the values of non-local variables.

Weakly-Invasive aspects may also change the values of the variables, as long as the returning state was already reachable in the underlying system.

Strongly-Invasive aspects are allowed to change the contents of the variables even when returning to states that were not originally reachable in the underlying system.

2.2 Temporal logic

Temporal logic allows expressing properties related to time formally. In particular, we work with linear temporal logic, which expresses for every computation path what must be satisfied. We will use propositional logic operators together with linear temporal logic (LTL) formulas. The operators available in LTL are

- $\mathbf{X}\varphi$: next φ. The next state must satisfy φ.
- $\mathbf{G}\varphi$: always φ. Every state in the future must satisfy φ.
- $\mathbf{F}\varphi$: eventually φ. There exists a state in the future that must satisfy φ.
- $\psi\mathbf{U}\varphi$: ψ until φ. Every state must satisfy ψ until φ holds.
- $\psi\mathbf{W}\varphi$: ψ weak-until φ. Equivalent to $(\mathbf{G}\psi) \vee (\psi\mathbf{U}\varphi)$

2.3 Weaving semantics

In this work we extend the ideas of interference detection in the sequential weaving model to the joint-weaving model.

The notation $S + A$ indicates the system S with A *woven into* it. A is aware of its joinpoints in S already, so $S + A$ results in the system where A is executed at all its joinpoints. However, in the sequential weaving $(S + A) + B$, B is woven into $S + A$, but if B has joinpoints of A, they are not recognized and the advice of A is not woven at those points.

In the sequential weaving model each aspect is woven to the system in a certain order, e.g. given the aspects A, B where there is no precedence defined among the aspects, the possible results are $(S + A) + B$ or $(S + B) + A$. Note that in both cases added joinpoints may not be recognized by the first woven aspect.

In the joint-weaving model, aspects may add or remove joinpoints of other aspects without restrictions, and all joinpoints are recognized. $S + (A, B)$ is used to denote this semantic model, which corresponds to the semantics of AspectJ.

2.4 Previous work

In MAVEN [8], every weakly-invasive aspect has to satisfy its specification (P, R), where P is the assumption and R the guarantee. A sequence of aspects $\{A_1, \ldots, A_n\}$ woven sequentially in this order is said to be interference-free if and only if whenever all assumptions are satisfied, and the aspects in the set are woven in that order, then all of the guarantees will be satisfied:

$$S \vDash \bigwedge_{i=1}^{n} P_{A_i} \;\Rightarrow\; ((S + A_1) + \ldots) + A_n \vDash \bigwedge_{i=1}^{n} R_{A_i}$$

In order to satisfy non-interference, every pair of aspects A, B is shown to satisfy two rules:

$\mathbf{KP_{AB}}$: For any system S that satisfies $P_A \wedge P_B$, the assumptions of both A and B, when A is woven into S the obtained system $(S + A)$ must preserve the assumption of B: $S \vDash P_A \wedge P_B \Rightarrow S + A \vDash P_B$.

$\mathbf{KR_{AB}}$: For any system S that satisfies the guarantee of A (R_A) and the assumption of B (P_B), when B is woven into S the obtained system $(S + B)$ must preserve the guarantee of A: $S \vDash R_A \wedge P_B \Rightarrow S + B \vDash R_A$.

In proving these properties, the state machine of the assumption (the tableau of the LTL formula) is used to represent all base programs satisfying the conjunction on the left hand side of the implication, and the appropriate aspect is woven to the state machine to yield a state machine that should satisfy the formula on the right hand side of the implication.

The rules are sufficient to guarantee correctness and non-interference only if (1) A cannot interfere with B's assumption while advice of B is executing (2) B cannot interfere with A's guarantee while A is executing, and (3) B does not in itself add new joinpoints of A. None of these conditions are true in our new general setting.

3. Specification

In this section we will consider the aspects given in Figure 1 to understand the specification method. Intuitively, `Auth` is correct if when woven to any system, every time it reaches a before `doTrans` joinpoint, eventually `Auth` returns where the value of `authenticated` is true if and only if the user and password exist in the system. This field can be used later to allow or not different actions on the transaction, for instance when not authenticated, the user may read but not write to the database.

The aspect `SaveCookie` is correct if when woven to any system, every time after the method `usrPwdExist` is completed and the user and password indeed exist, then a cookie is saved.

Finally, `EncryptPwd` guarantees that when woven, the aspect encrypts the password obtained by `requestPwd()`.

In the temporal logic representations of the specifications, for any aspect A, P_A represents the assumption on the underlying system and R_A expresses the guarantee of the augmented system when the aspect is woven. The formal specifications of the aspects are:

`Auth`**:**
$P_{\texttt{Auth}} = \texttt{true}$
$R_{\texttt{Auth}} = \mathbf{G}(doTrans_{before} \Rightarrow$
$(\mathbf{F}(usrRequested \wedge usr = U_0) \wedge$
$\mathbf{F}(pwdRequested \wedge pwd = P_0) \wedge$
$\mathbf{F}(ret_{\texttt{Auth}} \wedge authed \Leftrightarrow usrPwdInDB(U_0, P_0))))$

`SaveCookie`**:**
$P_{\texttt{SaveCookie}} = \texttt{true}$
$R_{\texttt{SaveCookie}} = \mathbf{G}((call\_usrPwdExist_{after} \wedge Success)$
$\Rightarrow \mathbf{F}\ cookieSaved)$

`EncryptPwd`**:**
$P_{\texttt{EncryptPwd}} = \texttt{true}$
$R_{\texttt{EncryptPwd}} = \mathbf{G}(call\_reqPwd \Rightarrow \mathbf{F}\ pwdEncrypted)$

The atomic propositions used for the aspects in Figure 1 are: $doTrans_{before}$, $call\_usrPwdExist_{after}$, $call\_reqPwd$ for every state which matches the respective aspect joinpoints, $ret_{\texttt{Auth}}$ to represent the return states of `Auth`, $authed$ represents the truth value of `authenticated` which may be used elsewhere in the system, the atomic proposition $usrPwdInDB$ indicates that the user and password exist in the database, $Success$ represents the returned Boolean value of the call to the method `usrPwdExist()`, and $cookieSaved$ represents that a cookie has been saved. Finally, $pwdEncrypted$ indicates that the password has been encrypted.

In the guarantee of `Auth`, U_0 and P_0 represent the input values, and can be thought of as bound to a universal quantifier. This can be expressed in propositional temporal logic by substituting each user and password pair (the domain is finite).

Note that for all three aspects, the guarantee has the form **G**(pointcut $\Rightarrow$ expected behavior). We will use this below to identify local and global guarantees.

4. Interference detection

4.1 Extended Specifications

If we attempt to apply the original rules to these aspects, problems arise. For example, although there is in fact no interference between `Auth` and `SaveCookie`, when the rule $KR_{\texttt{SaveCookie,Auth}}$ is checked, it will fail because it does not consider that `SaveCookie` will actually be executed at its new joinpoint added within `Auth`.

To treat the three conditions listed at the end of Section 2.4 that no longer hold (and thus are sources of inconsistency), we extend the specification of an aspect to include a distinction between global guarantees and local ones, and add an internal assumption.

Thus, we now consider guarantees of the form: $R = (RL, RG)$, where RL represents the local guarantee, a property that must be satisfied at each joinpoint, and RG a global guarantee, a global property not connected to being at a joinpoint.

The local guarantee can express properties both for each advice that starts executing because the current joinpoint matches the pointcut descriptor, or properties that should hold each time an advice of A has finished executing. Then, for an aspect A, RL is the conjunction of formulas of the form:

$\mathbf{G}(ptc_A \Rightarrow \varphi)$: Every time the pointcut of A is matched, φ should hold. Note that φ is not necessary a state property, but rather a temporal logic formula. φ is a formula expressing what A's execution guarantees.

In particular, guarantees of the form $\mathbf{G}(ret_A \Rightarrow \phi)$ expressing what is expected at the end of each execution of A can be translated to $\mathbf{G}(ptc_A \implies (\neg ret_A \mathbf{W}\ (ret_A \wedge \phi)))$ which has the form presented before: $(\varphi \equiv \neg ret_A \mathbf{W}\ (ret_A \wedge \phi))$.

This separation of the guarantee will be helpful in our verification, because each part is treated differently.

However, we still need to consider the difficulty that even if an aspect B does not interfere when activated before or after an aspect A, it might interfere *during* A's execution. To handle this, the specification is further extended to include internal assumptions that describe for each aspect A what is expected of aspects to be executed during A.

Thus, the specification of an aspect A now consists of:

external assumption of A (PE_A): The assumption about the system where A is to be woven. Equivalent to the previous P_A.

internal assumption of A (PI_A): The assumption on aspects to be executed during A.

local guarantee of A (RL_A): The guarantee of A true at each of its joinpoints.

global guarantee of A (RG_A): The part of the guarantee that must be satisfied even when A is not aware of all its joinpoints.

P_A is now defined as (PE_A, PI_A) and R_A is (RL_A, RG_A). An aspect specification is given by (P_A, R_A).

4.2 Partial Guarantee

In this section, we present the solution to the problem presented in Section 4.1.

In the given example the check of $KR_{\texttt{SaveCookie,Auth}}$ fails because there exist joinpoints of `SaveCookie` in $S + \texttt{Auth}$ that `SaveCookie` is not *aware* of (in particular, the one inside `Auth`).

In our extended aspect specification, we distinguished between the global guarantees and the local ones: those that express what must be satisfied at every place the advice is

executed. In our verification technique we consider only part of the joinpoints of A, but show that this is sufficient to guarantee correctness under full joint-weaving semantics.

In the model, we assume that weaving an aspect A to a system S adds a label aw_A to those states in S where A is aware of the joinpoint.

We now proceed to define a *partial guarantee* to express what must be satisfied when there are aspects not aware of every joinpoint. These do not follow the joint-weaving semantics, but are possible under sequential weaving.

Definition 1. *Let $RL_A = \bigwedge \psi_i$. For each ψ_i we define $\widetilde{\psi}_i = \mathbf{G}((ptc_A \wedge aw_A) \Rightarrow \varphi)$. Then the partial local guarantee of A is given by $\widetilde{RL}_A = \bigwedge \widetilde{\psi}_i$.*

$\widetilde{RL}_A$ is based on the local guarantee but including the atomic proposition that identifies whether an aspect is aware of a joinpoint, so it will be satisfied even when there are unaware joinpoints. When all aspects are aware of all their joinpoints, the original local guarantee is satisfied.

Definition 2. *The partial guarantee of A denoted $\widetilde{R}_A$ is given by $(\widetilde{RL}_A, RG_A)$.*

Note that for any system S, $S \vDash R_A \Rightarrow S \vDash \widetilde{R}_A$.

Moreover, if $S \vDash \widetilde{R}_A$ and A is aware of all its joinpoints in S, then $S \vDash R_A$.

Now, if the aspects of Figure 1 are checked taking the partial guarantee of `SaveCookie` ($\widetilde{R}_{\texttt{SaveCookie}}$) as in equation (1), every check is satisfied proving non-interference.

$$\widetilde{RL}_{\texttt{SaveCookie}} = \mathbf{G}((call\_usrPwdExist_{after} \wedge ret\_val = Success \wedge \underline{aw_{\texttt{SaveCookie}}}) \Rightarrow (Success = true \Rightarrow \mathbf{F}\ cookieSaved)) \quad (1)$$

However, this change in the rules is by itself unsound for joint-weaving semantics: when the aspects of Figure 1 are considered and checked against the partial guarantees, then every pairwise assertion is satisfied, returning that the set of aspects is apparently interference-free even though there is interference between `EncryptPwd` and `Auth`.

4.3 Augmented aspects

In order to present the full technique for verifying non-interference, the internal assumption must be considered, as seen in the following definition:

Definition 3. *An aspect A is* augmented *(noted as A^{+PI}) if and only if it has the internal assumption model woven into A.*

The augmented aspect is built by adding for each state that could be a joinpoint, a transition to the state machine that represents the internal assumption, and transitions from the final states of the internal assumption state machine to corresponding states of the aspect.

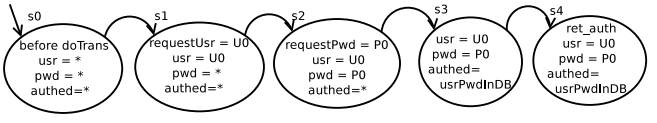

Figure 2. `Auth` model

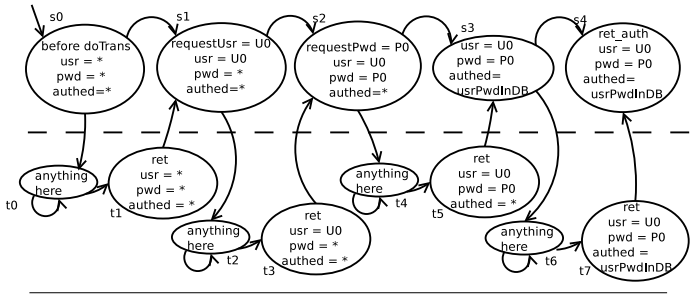

Figure 3. `Auth`$^{+PI}$ model

Example. A reasonable internal assumption for the aspect `Auth` in Figure 1 is for any inserted aspect to return without any exception thrown to the execution and preserve the values of the variables `usr`, `pwd` and `authenticated`. Hence, the aspect and the augmented version of the aspect are presented in Figures 2 and 3, respectively. Note that * in the figures represents some arbitrary initial value, different for each field.

The states $s_0, \ldots, s_4$ appear both in the aspect model and in the augmented model of the aspect.

In the augmented aspect, there is a transition from the state s_0 to the states t_0 and then t_1. This means that if an aspect is inserted at this point of `Auth` then the aspect can do anything (t_0) as long as eventually when reaching a returning state (t_1) the values of `usr`, `pwd` and `authenticated` are preserved, and hence it then executes the statement `Usr u = requestUsr()` of `Auth` (in state s_1).

It is assumed in this example that the fairness constraints are defined in order to avoid paths which stay infinitely in an "anything here" state.

Note that the augmented version of the aspect captures as well the assumption on aspects which share joinpoints, hence, shared-joinpoints interference is also covered by this compositional verification technique. In particular in the example, t_0 and t_1 represent the assumption of any aspect that may respond at the joinpoint `before doTrans()` and might be executed before the first statement of the aspect `Auth`.

4.4 Formal verification

In this section a modular verification technique to guarantee non-interference is presented, considering the extended specification with partial guarantees and internal assumptions.

First, the definition of non-interference for joint-weaving, based on that in [8], is presented.

Definition 4. *Let* $Aspects = \{A_1, \ldots, A_n\}$ *be a set of aspects. Let* (P_i, R_i) *be the specification of each aspect* A_i*. Then the set* $Aspects$ *is said to be* interference-free *if and only if* $OK_{Aspects}$ *holds.*

$$OK_{Aspects} \triangleq S \vDash \bigwedge_{i=1}^{n} P_i \Rightarrow S + (A_1, \ldots, A_n) \vDash \bigwedge_{i=1}^{n} R_i$$

$OK_{Aspects}$ expresses that weaving all aspects together with the joint-weaving model into any system S that satisfies the assumptions, satisfies the expected guarantees.

In [8] aspects were assumed not to add joinpoints of other aspects and hence the internal assumption was not necessary. The verification technique for each aspect on its own under the new specification simply incorporates the internal assumption by using the augmented aspect. Let A be the aspect to be verified and let (P_A, R_A) be A's specification. Then, A is correct with respect to its specification if and only if $S \vDash PE_A \Rightarrow S + A^{+PI_A} \vDash R_A$. This formula expresses that for every system S that satisfies the external assumption of A, when A is woven into S (with all its possible inserted aspects that satisfy the internal assumption) the resulting system satisfies both the local and global guarantees of R_A. Instead of verifying the aspect under the assumption that it is the only one, we verify it under the assumption (PI) that any other aspects maintain its correctness with respect to its specification.

When several aspects are jointly woven into a system S, all of them correct with respect to their specification, we intend to guarantee non-interference.

In order to achieve this, a set of rules is presented. If a library of aspects satisfies all the rules, then the library is interference-free, otherwise there may be interference.

The rules that aspects must satisfy in order to guarantee non-interference are now presented:

1. The aspect is correct by itself (OK_A^+):

   $$S \vDash PE_A \Rightarrow S + A^{+PI_A} \vDash R_A$$

   As explained above, this rule guarantees aspect correctness with respect to its specification. Given that the external assumption holds, the system obtained from weaving the aspect and all possibly inserted aspects must satisfy the local and global guarantees.

2. Considering A as the aspect currently being verified, and B any other aspect, the rules to detect interference are:

   KPI_{AB}^+**:** $S \vDash \widetilde{R}_A \wedge PE_B \Rightarrow S + B^{+PI_B} \vDash PI_A$

   This rule expresses that every aspect must satisfy the internal assumptions of other aspects.

   KPE_{AB}^+**:** $S \vDash PE_A \wedge PE_B \Rightarrow S + A^{+PI_A} \vDash PE_B$

   This rule expresses that when weaving A augmented to a system where the external assumption of another aspect B holds, this assumption should be preserved.

   KR_{AB}^+**:** $S \vDash \widetilde{R}_A \wedge PE_B \Rightarrow S + B^{+PI_B} \vDash \widetilde{R}_A$

   This rule expresses that when an aspect A has already been woven, weaving another aspect B preserves the partial guarantee of A (even if it adds new joinpoints of A).

In order to guarantee non-interference, rules KPI_{AB}^+, KPE_{AB}^+ and KR_{AB}^+ must be satisfied by every pair of aspects.

In the next sections we explain in more detail each rule.

4.4.1 KPI_{AB}^+

As mentioned above, rule KPI_{AB}^+ expresses that every aspect must satisfy the internal assumptions of other aspects.

The internal assumption of an aspect A determines what is expected of aspects that execute during A.

The general form of the internal assumption is (ρ, φ) where ρ is a propositional logic formula describing joinpoints and φ is a temporal logic formula describing restrictions on the behavior of the possible aspects executed at those joinpoints.

Example (Trans)**.** An aspect A may initiate a transaction and do some actions, and then close the transaction. We want to avoid that during the execution of the transaction any possibly woven aspect B may perform commit for that transaction. Then the internal assumption of A is defined as: any advice B to be executed at any state within the transaction of A should never perform commit until the return point is reached.

Such an internal assumption is given by the pair (ρ, φ) where $\rho : inTrans$ and $\varphi : \mathbf{G}\neg commit$.

Definition 5. *An augmented aspect* B^{+PI} *satisfies* A*'s internal assumption* (ρ, φ) *if and only if: for every execution* π *of* B *that starts from a state in* A *which satisfies* ρ *and matches* B*'s pointcut descriptor,* π *satisfies* φ*.*

Example. In Example (Trans), $B^{+PI} \vDash PI_A$ with $PI_A = (\rho, \varphi)$ as presented above if and only if for every joinpoint of B in A where A is in a transaction, B^{+PI} satisfies $\mathbf{G}\neg commit$.

Internal Assumption Defaults There are *default internal assumptions* that can be defined. Typical examples are:

$PI_A = NoAspect$**:** If the guarantee of the aspect A is sensitive to next state assertions (X) or real time constraints, PI_A may assume that no aspect is woven during A's execution.

$PI_A = Spectative$**:** It may be assumed from the environment that any aspect to be woven during the execution of A is spectative.

$PI_A = ReturningValuesPreserved\,(V)$**:** Perhaps, any woven aspect B may change things as long as when returning to the execution flow of A the values of a certain set of variables (V) remain as they were before executing

```
aspect LogDB: after call(send(msg))
  beforeStartTrans()
  startTrans()
  getTable(Log)->newRecord()
  getTable(Log)->setField(msgField, msg)
  getTable(Log)->setField(dateField, today)
  commit()
```

Figure 4. Internal assumption example

B. This is the internal assumption needed to preserve the values of `usr`, `pwd` and `authenticated` in `Auth`.

$PI_A = Invariant\,(I)$**:** Any aspect to be executed during A may need to satisfy a certain invariant I at every state.

$PI_A = ReturnsOK$**:** It can be assumed that every aspect executed within A terminates without throwing exceptions.

$PI_A = NoMandatoryProceed$**:** It can be assumed that by default all around advices have the proceed statement, but if aspects are allowed to be inserted into A without having to satisfy this condition (and hence possibly removing joinpoints), it can be identified by $NoMandatoryProceed$. This internal assumption differs from the previous ones in that instead of restricting the possible aspects, it allows more behaviors. The idea is that in most cases around advices still have $proceed$, and with this PI the particular cases in which there is no $proceed$ are considered.

Internal assumptions can be combined by overriding $\oplus$, conjunction $\wedge$ or disjunction $\vee$ of PI assumptions. Thus, combining a $ReturningValuesPreserved$ assumption with an internal assumption (ρ, φ) may look like: $PI = ReturningValuesPreserved\,(V) \circledast (\rho, \varphi)$ where $\circledast \in \{\oplus, \wedge, \vee\}$.

Example. In Figure 4, the aspect $LogDB$ exhibits the use of an internal assumption as explained above. In this case:

$$PI_{LogDB} = ReturningValuesPreserved\,(msg) \wedge (inTrans, \mathbf{G}\neg commit)$$

expresses that any other aspect B to be executed during $LogDB$ while in a transaction should not commit that transaction. The intersection of assumptions guarantees that as well every advice to be woven preserves the value of msg.

Checking that an aspect satisfies the internal assumptions of another aspect may involve model checking or syntactic checks, depending on the internal assumption.

Now, we present the satisfiability conditions of default internal assumptions.

Definition 6. *An augmented aspect B^{+PI} satisfies the default internal assumptions of A (PI_A) - noted as $B^{+PI} \models PI_A$ - if one of the following conditions hold:*

1. *There are no joinpoints of B in A.*
2. *If $PI_A = Spectative$ and there is a joinpoint of B in A, then all the possible augmented executions of B from joinpoints of A are spectative. In terms of temporal logic: $B^{+PI} \models \mathbf{G}\,(V = V_0)$ where V are all the variables that are not local to B and V_0 represents their original values before B is executed.*
3. *If $PI_A = ReturningValuesPreserved\,(V)$ and there is a joinpoint of B in A, then all possible augmented executions of B from joinpoints of A preserve the values of the variables in V at the returning state. In terms of temporal logic: $B^{+PI} \models \mathbf{G}\,(ret_B \Rightarrow V = V_0)$.*
4. *If $PI_A = Invariant\,(I)$ and there is a joinpoint of B in A, then all possible augmented executions of B from joinpoints of A satisfy the invariant at every state. In terms of temporal logic: $B^{+PI} \models \mathbf{G}I$.*
5. *If $PI_A = ReturnsOK$ and there is a joinpoint of B in A, then all possible augmented executions of B from joinpoints of A reach a returning state without throwing any exception. In terms of temporal logic: $B^{+PI} \models \mathbf{F}(ret_B \wedge \neg exception\_thrown)$*
6. *If $NoMandatoryProceed \notin PI_A$ and there is a joinpoint of an around advice B in A, then B should have a proceed statement for every execution path in the augmented model of B starting from joinpoints of A.*

Example. Considering the program listing in Figure 1 and the rule KPI^+_{AB}: the augmented version of `EncryptPwd` should satisfy the internal assumptions of `Auth` to prove that these aspects do not interfere. The specifications of the aspects are now extended to include the following internal assumptions:

`Auth`**:**
$PI_{\texttt{Auth}} = ReturnsOK \wedge$
$ReturningValuesPreserved(usr, pwd, authed)$
`EncryptPwd`**:**
$PI_{\texttt{EncryptPwd}} = Spectative$

The actual interference will be detected in this example when evaluating $KPI^+_{\texttt{Auth,EncryptPwd}}$: In this case `EncryptPwd` does not satisfy the internal assumption of `Auth` of preserving the value of the password.

4.4.2 KPE^+_{AB} and KR^+_{AB}

Rules KPE^+_{AB} and KR^+_{AB} are the extensions of the rules described in 2.4, now considering possibly inserted aspects and partial guarantees.

Rule KPE^+_{AB} expresses that when weaving A augmented to a system where the external assumption of another aspect B holds, this assumption should be preserved.

Rule KR^+_{AB} expresses that when an aspect A has already been woven, weaving another aspect B preserves the partial guarantee of A.

Moreover, given that the conditions for checking KPI^+_{AB} and KR^+_{AB} are the same, in certain cases both rules can be considered together. However, in several situations the

model is smaller when checking both properties separately. Note that even though the rules only imply that the partial guarantee is preserved, eventually all aspects will be aware of all their joinpoints, hence the partial guarantee will imply the guarantee.

4.5 Steps for each aspect added

If a set of aspects $\{A_1, \ldots, A_{n-1}\}$ has been proven to be correct with respect to their specification and without interference, when adding a new aspect A_n with specification $((PE_n, PI_n), (RL_n, RG_n))$, then the following properties should be checked:

1. Check that $OK^+_{A_n}$ holds.
2. Check that $KPI^+_{A_iA_n}, KPI^+_{A_nA_i}, KPE^+_{A_iA_n}, KPE^+_{A_nA_i}$, $KR^+_{A_iA_n}$, and $KR^+_{A_nA_i}$ are satisfied for all $1 \leq i \leq n-1$.

When building a library of n aspects we must do: n checks for the OK^+ rule, n^2 for each of the rules KPI^+, KPE^+ and KR^+. In several cases checking KPI^+ does not require model checking but perhaps uses static/syntactic analysis to detect joinpoints, check whether an aspect B satisfies an invariant or B is spectative. All these checks are done as the library is constructed and then a set of interference-free aspects can be used for any system that satisfies all aspect assumptions.

5. Justifying the rules

5.1 Assumptions

In this paper we treat weakly-invasive aspects [12], where control is returned after an advice execution to a state which existed in some execution of the original system. In [10], verification is shown for strongly-invasive aspects, by adding an assumption U about the base system states previously unreachable that now can occur in the woven system after aspect advice completes. A relatively complex modular verification technique is given that treats sequential weaving without joinpoints in advice. The treatment here can also be applied to that technique, both for each aspect on its own and for the rules to detect interference.

We also assume that the aspects treated are never activated under their own execution flow, i.e. there is no recursion. Allowing recursion introduces the problem of analyzing termination and liveness properties possibly affected.

These assumptions can often be checked by already existing techniques. In [3], dataflow techniques were presented to detect aspect categories. To guarantee no recursion a dependency graph can be built and analyzed to check that no aspect depends on itself.

5.2 Soundness proof

We now show the soundness of the rules in order to guarantee non-interference of a set of aspects that satisfies the necessary conditions.

First we prove how the augmented versions of the aspects satisfy the partial guarantees when their preconditions initially hold. Secondly we show that this also holds for the original aspects and the full guarantee when considering joint-weaving semantics and all aspects are woven.

Lemma 1. *Let $\{A_1, \ldots, A_n\}$ be a set of aspects such that for all of them the previous checks have been applied and all assertions have been proven to hold. Then, for any system S such that $S \vDash \bigwedge_{i=1}^{n} PE_i$, S with all the augmented aspects woven satisfies their partial guarantees, i.e.*

$$S + \left(A_1^{+PI}, \ldots, A_n^{+PI}\right) \vDash \bigwedge_{i=1}^{n} \widetilde{R}_i$$

Proof. By induction on the number of aspects in the set.

- Base case: When adding one aspect A to the system S which satisfies PE_A, from OK^+_A, $S + A^{+PI_A} \vDash R_A$. Then in particular, $S + A^{+PI_A} \vDash \widetilde{R}_A$
- Inductive step: We assume by inductive hypothesis that for any system S such that $S \vDash \bigwedge_{i=1}^{n-1} PE_i$, then $S + \left(A_1^{+PI}, \ldots, A_{n-1}^{+PI}\right) \vDash \bigwedge_{i=1}^{n-1} \widetilde{R}_i$ and we want to see that for any system S such that $S \vDash \bigwedge_{i=1}^{n} PE_i$, then $S + \left(A_1^{+PI}, \ldots, A_n^{+PI}\right) \vDash \bigwedge_{i=1}^{n} \widetilde{R}_i$.
  Given that $S \vDash \bigwedge_{i=1}^{n} PE_i$, then in particular, $S \vDash \bigwedge_{i=1}^{n-1} PE_i$ and by the inductive hypothesis

$$S + \left(A_1^{+PI}, \ldots, A_{n-1}^{+PI}\right) \vDash \bigwedge_{i=1}^{n-1} \widetilde{R}_i$$

  First, we need to see that A_n's assumption still holds. From $KP^+_{A_iA_n}$ the assumption of A_n is preserved as other aspects are woven to the system. Hence, $S + \left(A_1^{+PI}, \ldots, A_{n-1}^{+PI}\right) \vDash PE_n$.
  Then, when weaving A_n^{+PI} to $S + \left(A_1^{+PI}, \ldots, A_{n-1}^{+PI}\right)$, the conjunction $\bigwedge_{i=1}^{n-1} \widetilde{R}_i$ is preserved from $KR^+_{A_nA_i}$ and for those places where the A_n is woven in the execution of an aspect A_i, the correctness is preserved from $KPI^+_{A_iA_n}$ and $OK^+_{A_i}$: the paths added by A_n are already considered in A_i^{+PI} and satisfy the corresponding $\widetilde{R}_i$.
  Weaving A_n may add joinpoints of already woven aspects, but these paths are already considered in the augmented version of A_n, and due to $KPI^+_{A_n,A_i}$ and $OK^+_{A_n}$ the guarantee of A_n is also preserved.
  Therefore $S + \left(A_1^{+PI}, \ldots, A_n^{+PI}\right) \vDash \bigwedge_{i=1}^{n} \widetilde{R}_i$.

□

The lemma shows that if all the conditions hold then weaving all the augmented versions of the aspects is interference-free. The next theorem uses this lemma in order to prove that if we have established that all the augmented versions of the aspects are interference-free then, in particular, there is no interference when considering the resulting system with the (not augmented) aspects woven.

Theorem 1. *Let $\{A_1, \ldots, A_n\}$ be a set of aspects such that for all of them the previous checks have been applied and all assertions have been proven to hold. Then $\{A_1, \ldots, A_n\}$ is interference-free. That is, for any system S such that $S \vDash \bigwedge_{i=1}^{n} PE_i$, then S with all the aspects woven satisfies their guarantees, i.e.*

$$S + (A_1, \ldots, A_n) \vDash \bigwedge_{i=1}^{n} R_i$$

Proof. From Lemma 1, for any system S such that $S \vDash \bigwedge_{i=1}^{n} PE_i$ it holds that

$$S + \left(A_1^{+PI}, \ldots, A_n^{+PI}\right) \vDash \bigwedge_{i=1}^{n} \widetilde{R}_i$$

In particular, $S^{+}_{Aspects} = S + \left(A_1^{+PI}, \ldots, A_n^{+PI}\right)$ is an over-approximation of $S_{Aspects} = S + (A_1, \ldots, A_n)$. That is, every path in $S_{Aspects}$ is a path in $S^{+}_{Aspects}$. Given that all R_i are formulas in LTL, $S^{+}_{Aspects} \vDash \bigwedge_{i=1}^{n} \widetilde{R}_i \Rightarrow S_{Aspects} \vDash \bigwedge_{i=1}^{n} \widetilde{R}_i$. Moreover, given that all aspects are already woven, then all aspects are aware of all their joinpoints, and hence: $S_{Aspects} \vDash \bigwedge_{i=1}^{n} \widetilde{R}_i \Rightarrow S_{Aspects} \vDash \bigwedge_{i=1}^{n} R_i$ □

Theorem 1 shows that this procedure is sound under the given assumptions. However, it is not complete. In particular, modularity affects completeness: there could be sets of aspects which are interference-free but this cannot be shown with the assumptions and guarantees defined. That is, there may be two aspects A and B, both correct with respect to their specification and when woven together there is no interference, but the rules fail because the assumption or guarantee are not preserved in an intermediate state of building the augmented model.

The main advantages of this interference detection process is that it is modular, it provides flexibility to different external and internal assumptions, and is also used to prove the correctness and non-interference of collaborative aspects (described in the next section).

6. Cooperation

Cooperation is tightly related to modularity: an aspect A may assume the existence of an aspect B that takes care of certain functionality and then A can be shown to be correct.

6.1 Examples of cooperation

Following, two examples are presented to show different types of cooperation.

Example (Encrypt)**.** The aspect `EncryptPwd` that encrypts the password being sent from a registration form may assume the existence of another aspect (`CheckPwd`) that only allows sending passwords that satisfy some criteria, e.g. that the password includes a combination of numbers, lowercase and uppercase letters.

`CheckPwd`**:**
$PE_{\texttt{CheckPwd}} = \texttt{true}$
$R_{\texttt{CheckPwd}} = \mathbf{G}\,(to\_be\_sent \Rightarrow correct)$
`EncryptPwd`**:**
$PE_{\texttt{EncryptPwd}} = \mathbf{G}\,(to\_be\_sent \Rightarrow correct)$
$R_{\texttt{EncryptPwd}} = \mathbf{G}(to\_be\_sent \Rightarrow \mathbf{F}\,(sent \wedge encrypted))$

It is obvious that not necessarily every system satisfies the assumption of `EncryptPwd`, but if `CheckPwd` is also woven, and satisfies its specification, then the assumption of `EncryptPwd` holds.

Example (Copy)**.** An aspect (`Copy`) saves the objects of a certain class when necessary, trying initially to save them to a database and cooperating with another aspect (`CopyToFile`) when copying to the database fails. `CopyToFile` copies objects to an xml file. Either way, the objects are guaranteed to be saved.

`Copy`**:**
$PE_{\texttt{Copy}} = \texttt{true}$
$PI_{\texttt{Copy}} = EXISTS\_ASPECT$
$\quad \mathbf{G}((call(DB.saveObject) \wedge DBerror) \Rightarrow$
$\quad\quad \mathbf{F}\,savedObjectToFile)$
$R_{\texttt{Copy}} = \mathbf{G}(objectChanged \Rightarrow$
$\quad \mathbf{F}\,(savedObjectToDB \vee savedObjectToFile))$
`CopyToFile`**:**
$PE_{\texttt{CopyToFile}} = \texttt{true}$
$PI_{\texttt{CopyToFile}} = Spectative$
$R_{\texttt{CopyToFile}} = \mathbf{G}((objectChanged \wedge$
$\quad \mathbf{F}((call(DB.saveObject) \wedge DBerror)))$
$\quad\quad \Rightarrow \mathbf{F}\,savedObjectToFile)$

$EXISTS\_ASPECT$ represents the assumption that there **must** be an aspect satisfying the internal assumption.

The specification of `Copy` guarantees that when an object is changed, it is eventually copied, either to the database or to a file. `Copy` assumes ($PI_{\texttt{Copy}}$) the existence of an aspect that saves the object to a file if there is an error when trying to save an object to the database.

The specification of `CopyToFile` does in fact guarantee this.

In the first example one aspect helps establish the external assumption of another, while in the second, it helps to establish the internal assumption.

6.2 Formal verification of cooperation

The idea is that if there exists an order in which the joint-weaving model can be built such that all preconditions are eventually satisfied and there is no interference, then the whole system can be woven together under joint-weaving semantics and the set of aspects is interference-free.

Definition 7. *An aspect A is* augmented considering cooperation*, if it is augmented and it includes only the paths where the expected $EXISTS\_ASPECT$ assumptions are woven.*

Example. In the augmented version of Copy, all paths include the cooperation assumption of an aspect that on error saves to a file.

Definition 8. *A set of aspects* $\{A_1, \ldots, A_n\}$ *is cooperation inductive if they can be arranged in a sequence* $A_{i_1}, A_{i_2}, \ldots, A_{i_n}$ *such that for all* k, $1 \leq k < n$

$$S \vDash PE_{i_k} \Rightarrow S + A_{i_k}^{+PI} \vDash PE_{i_{k+1}} \quad (2)$$

Equation (2) *expresses that when the assumption of* A_{i_k} *holds, then when weaving the augmented model of* A_{i_k}*, the assumption of* $A_{i_{k+1}}$ *holds.*

Given a set of aspects, finding an cooperation inductive sequence of the aspects is a necessary condition in order to prove correctness and interference-freedom, otherwise an aspect assumption may not hold, and hence weaving it does not imply its guarantee.

This sequence is not to indicate an order in which aspects are woven or executed, but in order to guarantee that eventually all aspect's assumptions will be satisfied, and hence the aspects can be woven and their guarantees will hold.

The proof of soundness of the method consists of three parts:

1. Proving that weaving the augmented version of the first $k-1$ aspects leads to the assumptions of the k^{th} aspect to hold (by induction on the length of a cooperation inductive sequence of aspects).
2. Use the previous proof in order check that when all the augmented aspects are woven then their partial guarantees are proven to hold (by induction on the length of an cooperation inductive sequence of aspects).
3. Finally, given that all the augmented aspects are woven and their partial guarantees hold, it can be proven that when all aspects are woven (not augmented), their guarantees do in fact hold.

The proofs mentioned in 1 and 2 are very similar to the proof of Lemma 1. The proof mentioned in item 3 follows the same logic as in Theorem 1.

Example. Considering the previous examples, now the correctness of both EncryptPwd in Example $(Encrypt)$ and Copy in Example $(Copy)$ can be shown.

In Example $(Encrypt)$, it is enough to consider $A_{i_1} = A_{\texttt{CheckPwd}}$ and $A_{i_2} = A_{\texttt{EncryptPwd}}$.

In Example $(Copy)$, the expected internal assumption is woven to build the augmented version of the aspect Copy and then CopyToFile is shown to satisfy the internal assumption.

Note that by means of the cooperation proofs, and a subset of aspects that is intended to be woven in an application, the necessary cooperative aspects are found, either by those that are found before in the sequence of assumptions, or that are forced to exist by the keyword $EXISTS\_ASPECT$.

```
aspect Req&EncrPwd
after returning: enterUsr()
  enterPwd()
  encryptDES()

aspect DESSave
after returning: encryptDES()
  savePwd()

aspect AESEncr
void around: encryptDES()
  encryptAES()
```

Figure 5. Removed joinpoints

7. Removing Joinpoints

In this section, we show that the extended specification and verification also handle the removal of joinpoints of one aspect by another.

Example. In Figure 5 we show an example with removed joinpoints. It is easy to see that the aspect AESEncr removes joinpoints of DESSave. DES and AES are encryption algorithms.

In a system in which initially the DES encryption algorithm was used, the specification of the aspects could be given by:

Req&EncrPwd:

$PE_{\texttt{Req\&EncrPwd}} = \texttt{true}$

$PI_{\texttt{Req\&EncrPwd}} = EXISTS\_ASPECT$
$\quad \mathbf{G}(encryptedDES \Rightarrow \mathbf{F}savePwd)$

$R_{\texttt{Req\&EncrPwd}} = \mathbf{G}(enterUsr_{after} \Rightarrow$
$\quad \mathbf{F}(enterPwd \wedge \mathbf{F}(encryptedPwd \wedge \mathbf{F}savePwd)))$

DESSave:

$PE_{\texttt{DESSave}} = \texttt{true}$

$PI_{\texttt{DESSave}} = Spectative$

$R_{\texttt{DESSave}} = \mathbf{G}(encryptedDES \Rightarrow \mathbf{F}savePwd)$

AESEncr:

$PE_{\texttt{AESEncr}} = \texttt{true}$

$PI_{\texttt{AESEncr}} = Spectative$

$R_{\texttt{AESEncr}} = \mathbf{G}(call\_encryptDES$
$\quad \Rightarrow \mathbf{X}((\mathbf{G}\ \neg encryptedDES) \wedge \mathbf{F}(encryptedAES)))$

That is, Req&EncrPwd takes care of requesting a password and calling the encryption algorithm and assumes the existence of DESSave, an aspect that guarantees that eventually the encrypted password is saved. It is possible to see the cooperation in the assertion $PI_{\texttt{Req\&EncrPwd}}$. For now, we concentrate on the guarantee that the password must be saved (not necessarily encrypted).

However, due to a security problem, it is decided to create an aspect such that around every call to DES it uses now the AES encryption algorithm. The guarantee of AESEncr indicates that every time encryptDES() is called, it guarantees that no password is encrypted using DES ($\mathbf{G}\ \neg encryptedDES$), but now every call to encrypt the password is replaced by encryptAES(). Then,

when the aspect AESEncr is checked with other aspects to detect interference, given that AESEncr has no proceed and $NoMandatoryProceed \notin PI_{\texttt{Req\&EncrPwd}}$ the rule $KPI^{+}_{\texttt{Req\&EncrPwd,AESEncr}}$ is not satisfied.

This interference causes that the joinpoint of the call to encryptDES() is removed, and hence the password is no longer saved.

Note that even if $NoMandatoryProceed$ did belong to the definition of $PI_{\texttt{Req\&EncrPwd}}$, then when checking $OK^{+}_{\texttt{Req\&EncrPwd}}$ the problem would have been detected, as there would be paths in $\texttt{Req\&EncrPwd}^{+PI}$ where the call to the encryption would not be reachable and hence, the password would not be saved.

Thus, in both cases there is interference, and the interference-freedom checks, as expected, do not succeed.

The cooperation among Req&EncrPwd and DESSave shows our technique with removed joinpoints, but this example serves also to get a better understanding of interference detection. We now consider the following, perhaps more natural, specification of Req&EncrPwd:

$$PE_{\texttt{Req\&EncrPwd}} = \texttt{true}$$
$$PI_{\texttt{Req\&EncrPwd}} = ReturningValuesPreserved(pwd)$$
$$R_{\texttt{Req\&EncrPwd}} = \mathbf{G}(enterUsr_{after} \Rightarrow \mathbf{F}(enterPwd \wedge \mathbf{F}encryptDES))$$

If the set of aspects is checked for interference, the rule $KPI^{+}_{\texttt{Req\&EncrPwd,AESEncr}}$ fails again because of the absence of $NoMandatoryProceed$ in $PI_{\texttt{Req\&EncrPwd}}$. Detecting interference in this case is correct given that around advices without proceed() may cause the password not to be encrypted.

Furthermore, if $NoMandatoryProceed$ belonged to $PI_{\texttt{Req\&EncrPwd}}$, the rule $OK^{+}_{\texttt{Req\&EncrPwd}}$ would fail as it would no longer be guaranteed that after each joinpoint the password is eventually entered and encrypted.

8. Related work

Advice specification composition has been considered in [4], where aspects that may change the effective specification are called *assistants*. However, only method invocation joinpoints are considered and obliviousness is affected - accepted aspects must be specified in the specification, and dynamic context is not considered.

Besides MAVEN [8, 9], other previous work [1, 6, 7, 13, 15] has treated estricted forms of syntactic and/or semantic interference, considering in some cases disjoint joinpoints and in others shared ones.

Shared joinpoints interference has been considered in [11]. By answering some questions given in natural language regarding the expected behavior, an automatic extended version of the specification is built, and then, MAVEN can be used to detect interference. However, the method does not work for joint-weaving semantics.

The work in [14] discusses modular reasoning in aspect oriented programming. The concept of *aspect-aware interfaces* extends object and aspect interfaces to include global knowledge. In our work, we represent this global knowledge using specification: what is expected of the system and other cross-cutting concerns. These specifications serve to characterize each aspect without giving details of its implementation, and the technique presented allows checking aspect correctness, and non-interference even before the underlying system is completely programmed.

Aspects may be woven at joinpoints exposed by a module's signature in [2], and other joinpoints within the module are ignored. This affects obliviousness (a module must expose the places where an aspect may be woven in its signature) and does not consider joinpoints internal to a module.

In [13] the idea of internal assumptions is represented by Hoare-logic assertions that cross-cutting concerns must satisfy. This approach describes the acceptable state changes. In our approach we show that a general temporal logic formula or some syntactic check is satisfied instead of considering only a Hoare logic assertion where advice is woven and returns.

The work in [15] works with interfaces in temporal logic (CTL in their case), covers removed joinpoints due to the absence of proceed, but assumes that any advice restores the stack to the same state it had before the advice execution, not covering weakly-invasive aspects in general. To capture cascading advice, the states at which advice might apply must have an accurate interface. Knowing which are the states and what advice might apply affects obliviousness. This might also be a problem in our approach, especially for cooperation.

In [6] unification conflicts are detected, which may yield false positives when considering the problem of detecting semantic aspect interference.

In [17], aspect dependencies are found using as a base Reaching Definitions Data-flow Analysis [16]. These dependencies do not necessarily lead to semantic interference, possibly yielding false positives (i.e., it only detects cases of suspected interference), and the summary transfer functions imply analyzing a particular underlying system, instead of considering the aspects as an independent library.

9. Conclusions

Systems that work under the aspect paradigm usually include more than one aspect. It is important to check both that each aspect satisfies its specification and that the interaction of aspects does not lead to interference.

Existing work did not capture important cases of aspect composition and cooperation under joint-weaving semantics. Here, we have extended the verification technique to detect interference under joint-weaving semantics when aspects may insert or remove joinpoints of other aspects as

long as this does not create a recursive call stack to a certain aspect.

The assumption part of an aspect A's specification should now consider both assumptions of the system to which A is woven, and the aspects that may be woven in the execution of A. This gives a better understanding of an aspect: its specification now characterizes the environment where the aspect executes correctly.

We have presented a set of possible default internal assumptions as well as the possibility to define special internal assumptions, identifying the joinpoints and the temporal logic formulas that woven aspects must satisfy.

Moreover, adding aspect internal assumptions to aspect specification allows building a modular proof of non-interference among a set of aspects. The proof can be built once - when the library is built - and when interference-freedom is established, the aspects may be used for any system guaranteeing the necessary external assumptions.

The guarantee of the specification is now divided into local and global guarantee in order to represent global properties and properties related to the places where advice is woven. This separation aids in characterizing the proof obligations under joint-weaving semantics.

This same technique, considering internal assumptions and partial guarantees, allows extending the techniques to prove the correctness of cooperative aspects, both in the case an aspect is needed to satisfy the external assumptions of another one, or when the internal assumption of an aspect A forces an aspect to exist in order to guarantee A's correctness.

This verification technique exposes aspect interactions, dependencies and cooperation that can help AOP developers gain a deeper insight into the system under development.

References

[1] M. Aksit, A. Rensink, and T. Staijen. A graph-transformation-based simulation approach for analysing aspect interference on shared join points. In *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, AOSD '09, pages 39–50, New York, NY, USA, 2009. ACM.

[2] J. Aldrich. Open modules: A proposal for modular reasoning in aspect-oriented programming. In *In Workshop on foundations of aspect-oriented languages*, pages 7–18, 2004.

[3] Y. Alperin-Tsimerman and S. Katz. Dataflow analysis for properties of aspect systems. In *Proceedings of 5th Haifa Verification Conference, LNCS 6405*, 2009.

[4] C. Clifton and G. T. Leavens. Observers and assistants: A proposal for modular aspect-oriented reasoning. In *In FOAL Workshop*, 2002.

[5] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Păsăreanu, Robby, and H. Zheng. Bandera: extracting finite-state models from java source code. In *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, pages 439–448, New York, NY, USA, 2000. ACM.

[6] R. Douence, P. Fradet, and M. Südholt. A framework for the detection and resolution of aspect interactions. In *Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering*, GPCE '02, pages 173–188, London, UK, 2002. Springer-Verlag.

[7] P. E. A. Durr, T. Staijen, L. M. J. Bergmans, and M. Akşit. Reasoning about semantic conflicts between aspects. Technical Report TR-CTIT-05-73, Centre for Telematics and Information Technology University of Twente, Enschede, September 2005.

[8] M. Goldman, E. Katz, and S. Katz. Maven: modular aspect verification and interference analysis. *Form. Methods Syst. Des.*, 37:61–92, November 2010.

[9] E. Katz and S. Katz. Incremental analysis of interference among aspects. In *Proceedings of the 7th workshop on Foundations of aspect-oriented languages*, FOAL '08, pages 29–38, New York, NY, USA, 2008. ACM.

[10] E. Katz and S. Katz. Modular verification of strongly invasive aspects: summary. In *Proceedings of the 2009 workshop on Foundations of aspect-oriented languages*, FOAL '09, pages 7–12, New York, NY, USA, 2009. ACM.

[11] E. Katz and S. Katz. User queries for specification refinement treating shared aspect join points. SEFM '10, pages 73–82, Washington, DC, USA, 2010. IEEE Computer Society.

[12] S. Katz. Aspect categories and classes of temporal properties. *T. Aspect-Oriented Software Development I*, pages 106–134, 2006.

[13] R. Khatchadourian, J. Dovland, and N. Soundarajan. Enforcing behavioral constraints in evolving aspect-oriented programs. In *Proceedings of the 7th workshop on Foundations of aspect-oriented languages*, FOAL '08, pages 19–28, New York, NY, USA, 2008. ACM.

[14] G. Kiczales and M. Mezini. Aspect-oriented programming and modular reasoning. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 49–58, New York, NY, USA, 2005. ACM.

[15] S. Krishnamurthi and K. Fisler. Foundations of incremental aspect model-checking. *ACM Trans. Softw. Eng. Methodol.*, 16, April 2007.

[16] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[17] N. Weston, F. Taiani, and A. Rashid. Interaction analysis for fault-tolerance in aspect-oriented programming. In *In Proc. Workshop on Methods, Models, and Tools for Fault Tolerance*, pages 95–102, 2007.