Modularizing Crosscutting Concerns in Component-Based Systems

Antoine El-Hokayem[1], Yliès Falcone[1], and Mohamad Jaber[2]

[1] Univ. Grenoble Alpes, Inria, LIG, Grenoble, France firstname.lastname@imag.fr
[2] American University of Beirut, Beirut, Lebanon mj54@aub.edu.lb

Abstract. We define a method to modularize crosscutting concerns in the Behavior Interaction Priority (BIP) component-based framework. Our method is inspired from the Aspect Oriented Programming (AOP) paradigm which was initially conceived to support the separation of concerns during the development of monolithic systems. BIP has a formal operational semantics and makes a clear separation between architecture and behavior to allow for compositional and incremental design and analysis of systems. We thus distinguish local from global aspects. Local aspects model concerns at the component level and are used to refine the behavior of components. Global aspects model concerns at the architecture level, and hence refine communications (synchronization and data transfer) between components. We formalize global aspects as well as their integration into a BIP system through rigorous transformation primitives and overview local aspects. We present `AOP-BIP`, a tool for Aspect-Oriented Programming of BIP systems, and demonstrate its use to modularize logging, security, and fault-tolerance in a network protocol.

1 Introduction

A component-based approach [2] consists in building complex systems by composing components (building blocks). This confers numerous advantages (e.g., productivity, incremental construction, compositionality) that allow to deal with complexity in the construction phase. Component-based design is based on the separation between coordination and computation. The isolation of coordination mechanisms allows a global treatment and analysis on coordination constraints between components even if local computations on components are not visible (i.e., components are "black boxes").

A typical system consists of its main logic along with tangled code that implements multiple other functionalities. Such functionalities are often seen as secondary to the system. For example, logging is not particularly related to the main logic of most systems, yet it is often scattered throughout multiple locations in the code. Logging and the main code are separate domains and represent different *concerns*. A concern is defined in [5] as a "*domain used as a decomposition criterion for a system or another domain with that concern*". Domains include logging, persistence and system policies like security. Concerns are often found in different parts of a system, or in some cases multiple concerns overlap one region. These are called *crosscutting concerns*. AOP aims at modularizing crosscutting

concerns by identifying a clear role for each of them in the system, implementing each concern in a separate module, and loosely coupling each module to only a limited number of other modules. At a glance, AOP defines mechanisms to determine the locations of the concerns in the system execution by introducing the concept of joinpoints and pointcuts. Then, it determines what to do at these locations by introducing advices. Finally, it provides a mechanism to coordinate all advices happening at a location by introducing a process called weaving.

Motivations and challenges. In CBSs, crosscutting concerns arise at the levels of components [9, 18] (building blocks) and architectures (communications). Integrating crosscutting concerns in CBSs improves the progressiveness of building complex systems. More importantly, it allows users to reason about crosscutting concerns in separation, and favors correct-by-construction design.

Defining an AOP paradigm for CBSs poses multiple challenges. Firstly, the notion of program execution, while clear in a sequential program, needs to be redefined for CBSs. Indeed, the execution of a sequential program can be seen as a sequence of instructions, whereas the semantics of a CBS is generally more complex and relies on a notion of architecture imposing several constraints on their execution. Secondly, we aim to ensure that the locations where concerns arise in CBSs are represented homogeneously. This facilitates the verification and instrumentation of the system when incorporating crosscutting concerns (both at the syntactic and semantic levels). Finally, at any location, it is necessary to identify the possible modifications of a CBS that preserve semantics and coordination constraints.

Approach. We use the Behavior Interaction Priority (BIP) component-based framework [2, 19] with formal operational semantics. Coordination between components is achieved by using multiparty interactions and dynamic priorities for scheduling interactions. BIP consists of three layers: (1) Behavior which is handled by atomic components; (2) Interaction that describes the collaboration between the atomic components; (3) Priority chooses which interaction to execute out of many. BIP can be used to formally specify CBSs and generate efficient code that implements a CBS description.

We augment the BIP framework with the aspect-oriented paradigm. We begin by presenting the concepts of the BIP framework in Sec. 2. In general, concerns are expressed by determining their locations in the system, and their behavior at the given locations. Based on the formalization of concerns, we determine the *rules* that govern the integration of these concerns in a BIP system. Therefore, given an initial BIP system, and a description of concerns, we transform it so as to include the desired concerns. We distinguish and define two types of aspects: *Global* and *Local*. In Sec. 3, we give a full definition of global aspects and we briefly, for the lack of space, discuss local aspects and the composition of aspects. A full description of (1) local aspects, (2) aspect containers (which serve as a construct for composing aspects), (3) a high-level language for writing local and global aspects; and (4) the full example can all be found in [10]. Sec. 4 describes the `AOP-BIP` tool and its evaluation on network protocol case study. We present related work in Sec. 5 and future work in Sec. 6.

2 Behavior Interaction Priority

Behavior Interaction Priority (BIP) [2, 19] allows to define systems as sets of atomic components with prioritized interactions. We present components, interactions, priorities, and their composition. An atomic component is the basic computation unit. It is defined by its interface (i.e., a set of ports) and behavior defined as a Labeled Transition System (LTS) extended with data. Transitions are labeled with update functions, guards, and ports. Ports define communication and synchronization points for components. A port can be associated with some variables (of the component), to exchange data with other components. Ports are said to be exported by the component as they define its interface.

Definition 1 (Update function). *An update function over a set of variables X is a sequence of assignments $\langle x_1 := f^1(X_1), \ldots, x_n := f^n(X_n)\rangle$, where $\forall i \in [1, n] : x_i \in X \wedge X_i \subseteq X$.*

Definition 2 (Port). *A port $\langle p, x_p\rangle$ is defined by an identifier p and a set of attached local variables x_p (denoted by $p.vars$).*

Definition 3 (Atomic component). *An atomic component is a tuple $\langle P, L, T, X\rangle$, where:*
- *X is a set of variables.*
- *L is a set of control locations.*
- *P is the set of ports such that $\forall p \in P : p.vars \subseteq X$.*
- *$T = L \times P \times \mathbb{B}[X] \times Exp[X] \times L$ is the set of transitions, where $\mathbb{B}[X]$ (resp. $Exp[X]$) is the set of boolean predicates (resp. update functions) over X.*

In a transition $\tau = \langle \ell, p_\tau, g_\tau f_\tau, \ell'\rangle \in T$, (1) ℓ is the source location; (2) ℓ' is the destination location; (3) p_τ is a port exported by the component; (4) g_τ is the guard (a boolean predicate), a boolean function over X; (5) f_τ is an update function over X. For a component $B = \langle P, L, T, X\rangle$ we denote P, L, T, X, by $B.locs$, $B.ports$, $B.trans$, $B.vars$, respectively. Additionally, we denote by $\mathcal{B}$ the set of all atomic components. Furthermore, for a transition $\tau = \langle \ell, p, g, f, \ell'\rangle$, we denote ℓ, p, g, f, ℓ' by $\tau.src$, $\tau.port$, $\tau.guard$, $\tau.func$, $\tau.dest$, respectively.

The semantics of an atomic component B is defined as an LTS. A state of the LTS consists of a location and valuation v of the variables of B. A transition is labeled with port along with valuation of its variables v_p, which is possibly received from other components. A transition $\langle l, p[X_p], g_\tau, f_\tau, l'\rangle$ is possible iff B has a transition $\tau = \langle \ell, p[X_p], g_\tau, f_\tau, \ell'\rangle \in T$ such that: (1) the guard before receiving the new valuation v_p of the port variables holds, i.e., $g_\tau(v) = true$; (2) the application of the computation function $f_\tau(v_p/v)$ yields v'.

Definition 4 (Semantics of an atomic component). *The semantics of an atomic component B is the LTS $S_B = \langle B.locs \times \boldsymbol{X}, B.ports \times \boldsymbol{X}, \rightarrow\rangle$, where: $\rightarrow = \{\langle\langle l, v\rangle, p(v_p), \langle l', v'\rangle\rangle \mid \exists \tau = \langle l, p[X_p], g_\tau, f_\tau, l'\rangle \in T.trans : g_\tau(v) \wedge v' = f_\tau(v_p/v)\}$; and, $\boldsymbol{X}$ denotes the set of possible valuations of the variables in X.*

Furthermore, we say that a port p is enabled in a state $\langle \ell, v\rangle$, if there exists at least one transition τ from ℓ labeled by p and its guard $g_\tau(v)$ holds.

Interactions serve as the glue that coordinates (i.e., synchronization and data transfer) the components through their ports. An interaction consists of one or more ports of different atomic components, a guard on the variables of its ports, an update function that realizes data transfer between the ports.

Definition 5 (Interaction). *An interaction a is a tuple $\langle P_a, F_a, G_a \rangle$ s.t.:*
- *$P_a \subseteq \bigcup_{B \in \mathcal{B}}(B.ports)$ is a nonempty set of ports not containing more than one port per atomic component, i.e., $\forall B \in \mathcal{B} : |B.ports \cap P_a| \leq 1$.*
- *F_a is an update function over $\bigcup_{p_i \in P_a}(p_i.vars)$ executed with the interaction.*
- *G_a is a boolean expression, the guard of the interaction.*

For an interaction a, we denote P_a, G_a, F_a, as $a.ports$, $a.guard$, $a.func$ respectively.

We fix $\mathcal{B} = \{B_1, \ldots, B_n\}$ as the set of atomic components where the semantics of B_i is $S_{Bi} = \langle Q_{Bi}, P_{Bi}, \rightarrow \rangle$, $i \in [1, n]$, and γ as the set of interactions. A composite component is defined by composing atomic components using glue consisting of interactions and priorities.

Definition 6 (Semantics of composite component). *The semantics of the composite component built with $\mathcal{B}$ and γ (noted $\gamma(\mathcal{B})$) is the LTS $\langle Q, \gamma, \rightarrow \rangle$ where $Q = Q_{B_1} \times Q_{B_2} \times \ldots \times Q_{B_n}$, and $\rightarrow$ is the least set of transitions satisfying*

$$\frac{a = (\{p_i\}_{i \in I}, G_a, F_a) \in \gamma \qquad G_a(\{v_{pi}\}_{i \in I}) \qquad \forall i \in I, q_i \xrightarrow{p_i(v_i)}_i q_i' \wedge v_i = F_a^i(\{v_{pi}\}_{i \in I}) \qquad \forall i \notin I, q_i = q_i'}{\langle q_1, \ldots, q_n \rangle \xrightarrow{a} \langle q_1', \ldots, q_n' \rangle}$$

where v_{pi} is the valuation of the variables attached to port p_i and F_a^i is the partial update function derived from F_a restricted to the variables of p_i.

An interaction a is enabled iff its guard G_a holds and all of its ports are enabled. An enabled interaction is selected from the complete list of interactions, based on the current states of the atomic components. The BIP engine selects one of the enabled interactions and executes its update function F_a, which may modify its port variables. Then, the involved atomic components execute their corresponding transitions given the new valuations v_i received by the selected ports. In the following, we consider a composite component $\mathcal{C} = \gamma(\mathcal{B})$ with behavior $\langle Q, \gamma, \rightarrow \rangle$.

Multiple interactions can be enabled in a configuration. Priorities are used to filter the enabled interactions and reduce non-determinism.

Definition 7 (Priority). *A priority model π over $\mathcal{C}$ is a strict partial order on the set of interactions γ. We abbreviate $\langle a, a' \rangle \in \pi$ by $a \prec_\pi a'$. Adding π to $\mathcal{C}$ results in a new composite component $\mathcal{C}' = \pi(\mathcal{C})$ which semantics is the LTS $\langle Q, \gamma, \rightarrow_\pi \rangle$ where $\rightarrow_\pi$ is the least set of transitions satisfying the following rule:*

$$\frac{q \xrightarrow{a} q' \qquad \neg(\exists a' \in \gamma, \exists q'' \in Q : a \prec_\pi a' \wedge q \xrightarrow{a'} q'')}{q \xrightarrow{a}_\pi q'}$$

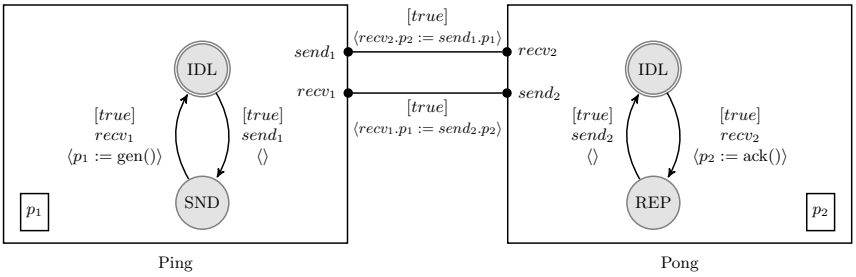


Fig. 1: Two communicating agents

Whenever according to π an interaction $a \in \gamma$ is selected, there does not exist an enabled interaction in γ which has higher priority than a.

A composite component obtained by the composition of a set of atomic components can be composed with other components (composite or atomic) in a hierarchical and incremental fashion using the same operational semantics. For the scope of this paper, we flatten a hierarchical composite component to obtain a non-hierarchical one (i.e., consisting only of atomic components and simple interactions) using the method presented in [4]. The non-hierarchical composite component is subsequently referred to as the BIP model. A BIP system is constructed by composing atomic components using interactions and priorities (to form the BIP model), with an initial state (initial locations and variable initialization of atomic components).

Definition 8 (BIP system). *A BIP system is a tuple $\langle \mathcal{C}, q_0 \rangle$, where $q_0 = \langle Init, v \rangle$ is the initial state with $Init \in B_1.locs \times \ldots \times B_n.locs$ being the tuple of initial locations of atomic components, and $v \in \boldsymbol{X}^{Init}$ is the tuple formed by the initial valuations of all variables in atomic components $X^{Init} \subseteq \bigcup_{B \in \mathcal{B}}(B.vars)$.*

Example 1 (BIP System). Figure 1 depicts a BIP system composed of two atomic components: `Ping` and `Pong`. The `Ping` component has one variable p_1 initialized to a random number and two locations `IDL` and `SND`, and two ports $send_1$ and $recv_1$. We associate the variable p_1 with both $send_1$ and $recv_1$. Initially, the `Ping` and `Pong` components are at the `IDL` locations. From location `IDL`, in component `Ping`, port `send1` is enabled, since the guard of the transition from `IDL` to `SND` holds. Similarly the transition from `IDL` to `REP` in `Pong` is possible, and $recv_2$ is enabled. The interaction that has both ports $send_1$ and $recv_2$ enabled, and its guard holding, is now enabled. Since no other interaction is enabled, it executes. Its update function executes the data transfer using the ports $send_1$ and $recv_2$ and their associated variables p_1 and p_2. Then, the update function of each transition executes (generating the acknowledgment packet in `Pong`). `Ping` will move to location `SND` and `Pong` to `REP`. Similarly, on the next step, the acknowledgement is sent back to `Ping` and it generates a new number. The two interactions ensure synchronization between the two components.

3 Modularizing Crosscutting Concerns in BIP

We address the concerns arising in the global view, namely the view where atomic components are black boxes and only the interactions are accessible.

3.1 Preliminaries

For the rest of the paper, we fix an arbitrary BIP-system $\langle \mathcal{C}, q_0 \rangle$ where $\mathcal{C} = \pi(\gamma(\mathcal{B}))$ with semantics $S = \langle Q, \gamma, \rightarrow \rangle$.

We abstract the execution of a BIP system as a trace. Then, we define operations for inspecting data access and control flow.

Definition 9 (BIP trace). *A BIP trace $\rho = (q_0 \cdot a_0 \cdot q_1 \cdot a_1 \cdots q_{i-1} \cdot a_{i-1} \cdot q_i)$ is an alternating sequence of states of S and interactions in γ; and $q_k \xrightarrow{a_k} q_{k+1} \in \rightarrow$, for $k \in [0, i-1]$.*

A global event defines an interaction execution moving the system from a state to another state. From a trace, one can extract a sequence of global events.

Definition 10 (Global events). *The sequence of global events E_ρ extracted from the trace $\rho = (q_0 \cdot a_0 \cdot q_1 \cdots a_{i-1} \cdot q_{i-1} \cdot a_{i-1} \cdot q_i)$ is $(E_0 \cdot E_1 \cdots E_{i-1})$ where $E_k = \langle q_k, a_k, q_{k+1} \rangle$, for $k \in [0, i-1]$.*

Definition 11 (readvar **and** writevar**).** *Given a set of variables X and an update function $f = \langle x_1 := f^1(X_1), \ldots, x_n := f^n(X_n) \rangle$ as per Definition 1:*
- readvar$(f) = X_1 \cup X_2 \cup \ldots \cup X_n$ *denotes the* read *variables;*
- writevar$(f) = \{x_1, x_2, \ldots, x_n\}$ *denotes the* modified *variables.*

3.2 Global Joinpoints

In the global view, we focus on the atomic components used in a composite component. These components only export their ports, on which interactions are defined. Generally, each atomic component computes its enabled ports. Given the enabled ports and the guards of the interactions, the composite component executes one interaction which has: (1) all its ports enabled, (2) its guard holds, (3) there does not exist another interaction with higher priority which is also enabled. At the interaction level, the following operations exist: interaction enablement[3] and interaction execution.

Whenever an interaction executes, three kinds of global joinpoints can be identified: (1) Synchronization between different atomic components; (2) One or more atomic components sending data; (3) One or more atomic components receiving data. In the case of the global view, a joinpoint is simply any event appearing in the execution (in the sense of Definition 10). For the rest of the section, we consider $\mathcal{E}$ to be the set of all reachable events in $\langle \mathcal{C}, q_0 \rangle$ with $\rightarrow$.

Definition 12 (Global joinpoint). *A global joinpoint is a global event $E \in \mathcal{E}$.*

3.3 Global Pointcuts

Since we only consider the interaction execution joinpoint, we consider criteria for matching interactions and relate them to global joinpoints. To select

[3] In this paper, we only consider *interaction execution* because of the complexity of matching *interaction enablement* which requires to include the BIP engine as part of the BIP model. To consider interaction enablement, it is better to interface with the BIP engine, meaning re-implement the BIP runtime.

a set of interactions, we use constraints over their associated ports (and their variables) and the involved data transfer. For this, a global pointcut expression has three parts: the ports themselves, a set of read variables, and a set of write variables. Note that the port variables should be involved in the computation function of the interaction. In Sec. 3.4, we use the read and written variables to define the context information passed to the advice.

Definition 13 (Global pointcut). *A global pointcut is a 3-tuple* $\langle p, v_r, v_w \rangle$ *that satisfies the following constraints:*
- $p \subseteq \bigcup_{B \in \mathcal{B}}(B.ports)$ *is a set of ports.*
- $v_r \subseteq \bigcup_{p_i \in p}(p_i.vars)$ *is the set of* read *variables.*
- $v_w \subseteq \bigcup_{p_i \in p}(p_i.vars)$ *is the set of* modified *variables.*

Definition 14 (Matching a global joinpoint with a global pointcut). *A global event* $\langle q, a, q' \rangle$ *is a joinpoint selected by a global pointcut* $\langle p, v_r, v_w \rangle$ *iff* $\langle q, a, q' \rangle \vDash \langle p, v_r, v_w \rangle$*, where:*

$$\langle q, a, q' \rangle \vDash \langle p, v_r, v_w \rangle \text{ iff } p \subseteq a.ports \wedge v_r \subseteq \text{readvar}(a.func) \wedge v_w \subseteq \text{writevar}(a.func).$$

A global event $\langle q, a, q' \rangle$ matches a global pointcut $\langle p, v_r, v_w \rangle$ if the interaction a involves all the ports in p, and its update function reads from the variables in v_r and writes to the variables in v_w.

A global pointcut captures interaction execution. For this purpose, we capture the interactions on the syntax of BIP models. Matching a global pointcut consists in selecting a subset of the interactions of a composite component.

Proposition 1. $(\langle q, a, q' \rangle \vDash gpc)$ *iff* $a \in \text{match}_{\text{g}}(\mathcal{C}, gpc)$*, where:* $\text{match}_{\text{g}}(\mathcal{C}, \langle p, v_r, v_w \rangle) = \{a' \in \gamma \mid p \subseteq a'.ports \wedge v_r \subseteq \text{readvar}(a'.func) \wedge v_w \subseteq \text{writevar}(a'.func)\}$

The proposition ensures that an event is a joinpoint (i.e., $\langle q, a, q' \rangle \vDash gpc$)) iff its interaction a is syntactically selected (i.e., $a \in \text{match}_{\text{g}}(\mathcal{C}, gpc)$).

Example 2 (Interactions matched by a pointcut). Figure 2a shows the interactions obtained by matching four pointcuts:

1. $\langle \{pa_1, pb_1\}, \emptyset, \emptyset \rangle$ matches all interactions including $\{pa_1, pb_1\}$ in their ports, that is, it only matches a_0 as it is the only interaction involving both ports.
2. $\langle \{pb_2\}, \emptyset, \emptyset \rangle$ matches all interactions including $\{pb_2\}$ in their ports, that is, it matches interactions a_1 and a_3, since they both involve pb_2.
3. $\langle \{pb_2\}, \{x_b\}, \emptyset \rangle$ matches interactions including $\{pb_2\}$ and which computation reads variable x_b associated with pb_2. The pointcut only matches a_1.
4. $\langle \{pd_1\}, \{x_d\}, \{x_d\} \rangle$ matches interactions that include $\{pd_1\}$ and which computation read and write the variable x_d associated with pd_1 (to receive data). The pointcut only matches a_1.

3.4 Global Advice and Global Aspect

A global advice defines the possible actions allowed on a global joinpoint $\langle q, a, q' \rangle$. These actions are restricted to two update functions f_b and f_a respectively before and after the interaction function $a.func$. Moreover, a global

advice can only modify the ports that it matches, as interactions could include other ports. The non-matching ports and their variables are hidden from the advice as per application of Demeter's law [17]. Given a global pointcut $pc = \langle \{p_1, \ldots, p_n\}, v_r, v_w \rangle$, an advice is restricted to the ports $\{p_1, \ldots, p_n\}$ and their variables, and a set of extra variables V called the inter-type variables.

Definition 15 (Global advice). *Given a set of ports $p \subseteq \bigcup_{B \in \mathcal{B}} B.ports$ and a set of inter-type variables V, $X_{\text{adv}} = V \cup \bigcup_{p_i \in p}(p_i.vars)$ is the set of advice variables. A global advice is a pair of functions $\langle f_b, f_a \rangle$ such that:*
- $(\text{readvar}(f_b) \cup \text{writevar}(f_b)) \subseteq X_{\text{adv}}$, and
- $(\text{readvar}(f_a) \cup \text{writevar}(f_a)) \subseteq X_{\text{adv}}$.
The global advice bound to p and V is noted $gadv(p, V)$.

The functions f_b and f_a are referred to as the before and after advice functions, respectively. The variables that they read and write (captured with readvar and writevar, respectively) should be variables of the advice.

We bind an advice to a pointcut expression with a global aspect. The advice should then apply to every joinpoint that the pointcut matches.

Definition 16 (Global aspect). *A global aspect is a tuple $\langle \mathcal{C}, V, gpc, gadv(p, V) \rangle$ such that:*
- $\mathcal{C}$ is a composite component (as per Definition 8);
- V is the set of variables associated with the aspect;
- $gpc = \langle p, v_r, v_w \rangle$ is the global pointcut (as per Definition 13);
- $gadv(p, V)$ is the global advice (as per Definition 15).

A global aspect therefore acts as a constraint between the pointcut and the advice. It ensures that the ports referred to the pointcut are the same for the advice, and that the advice has access to the variables of all ports in p and V.

Example 3 (Global Advice and Global Aspect). The global aspect:

$$\langle \mathcal{C}, \{v_0\}, \langle \{pd_1\}, \{x_d\}, \{x_d\} \rangle, \langle v_0 := x_d, x_d := v_0 \rangle \rangle$$

defines the inter-type variable v_0. It also defines the pointcut to match the interactions that include port pd_1 and which update function reads and writes to x_d. The advice's before and after update functions are respectively $\langle v_0 := x_d \rangle$ and $\langle x_d := v_0 \rangle$; saving the value of x_d in v_0 before the update function executes and then setting it back afterwards. The pointcut matches a_1 as shown in Fig. 2a and specifies that a_1 should execute the following sequence of instructions: $\langle (v_0 := x_d), (x_d := x_d + x_b), (x_d := v_0) \rangle$. An advice function in this case can only access $\{v_0\} \cup pd_1.vars$. The advice functions have no access to x_b, as it is not related to the port pd_1 but pb_2. This aspect disallows all interactions that read and write to x_d to modify its value.

3.5 Global Weaving

Using the binding of an advice to a pointcut, the weaving procedure instruments the BIP model. The procedure ensures that whenever a joinpoint matched

by a pointcut occurs, the BIP system executes the advice. Recall that interactions are stateless (i.e., they have no variables of their own), but they rely on data transfer from ports. Variables can only be defined in atomic components. Therefore, the weaving procedure must create an extra atomic component (so called inter-type component) that contains the variables of the aspect along with necessary ports and interactions. The weaving operation is only concerned with syntactically modifying the system representation. For this, we separate the two notions of matching to find the locations to modify from the instrumentation itself. We therefore define first the transformation procedure and then its application with matching.

The transformation procedure uses the following parameters:

- A BIP composite component $\mathcal{C}$;
- A set of interactions $\mathcal{I}$ resulting from syntactically matching a pointcut;
- A set of extra variables (i.e., the inter-type variables);
- The two functions f_b and f_a of the advice.

Accordingly, we create a new BIP composite component where the update function of each $a \in \mathcal{I}$ is preceded by f_b and followed by f_a. In the following, we describe the weaving of a global aspect which requires weaving of the inter-type component and weaving of the advice.

Generating an inter-type component. We first define the inter-type component.

Definition 17 (Inter-type component). *The inter-type component associated to the set of inter-type variables V is defined as $B_V = \langle \{p_V\}, \{\ell_0\}, \{\langle \ell_0, p_V, true, \langle\rangle, \ell_0 \rangle\}, V \rangle$ where $p_V = \langle p_V, V \rangle$.*

B_V contains V as its variables, one port $p_V = \langle p_V, V \rangle$ with all the variables attached to it, and one control location with a transition labeled with p_V and guarded with the expression $true$. This ensures that the port will not stop any other interaction from executing once connected to it. The inter-type component is added to the set of atomic components $\mathcal{B}$ of the BIP system.

Example 4 (Adding an inter-type component to a system). Figure 2b depicts $\mathcal{C}' = \pi(\gamma(\mathcal{B} \cup \{B_V\}))$ where $V = \{v_0, v_1\}$ and $\mathcal{C} = \pi(\gamma(\mathcal{B}))$. A new atomic component B_V is created. B_V has two local variables v_0 and v_1 and has its port p_V always enabled. The variables in V are attached to p_V.

Weaving the advice. Once the inter-type component is added to the system, the advice is woven by connecting the existing interactions to it.

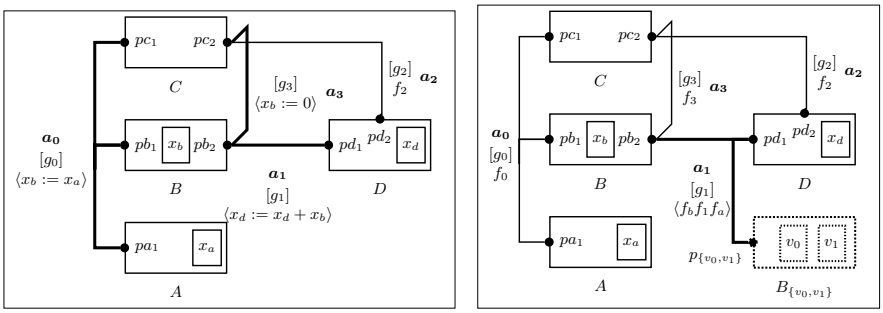
Definition 18 (Global weave). *Given a composite component $\mathcal{C} = \pi(\gamma(\mathcal{B}))$, a set of interactions $\mathcal{I}$, an inter-type V, and a global advice $adv = \langle f_b, f_a \rangle$, the global weave is defined as $\mathcal{C}' = \text{weave}_{\text{g}}(\mathcal{C}, \mathcal{I}, V, adv)$ where $\mathcal{C}' = \pi(\gamma'(\mathcal{B}'))$ is the new composite component; with:*

- *$\mathcal{B}' = \mathcal{B} \cup \{B_V\}$ is the new set of atomic components;*
- *$B_V = \langle \{p_V\}, \{\ell_0\}, \{\langle \ell_0, p_V, true, \langle\rangle, \ell_0 \rangle\}, V \rangle$ is the inter-type component identified by V (as per Definition 17);*

– γ' *is defined as* $\{\mathrm{m}(a) \mid a \in \gamma\}$ *with:*

$$\mathrm{m}(a) : \gamma \to \gamma' = \begin{cases} \langle a.ports \cup \{p_V\}, f_b \cdot a.func \cdot f_a, a.guard \rangle & \text{if } a \in \mathcal{I}, \\ a & \text{otherwise.} \end{cases}$$

The inter-type component B_V is added to $\mathcal{B}$. The interactions that require instrumentation (i.e., $a \in (\gamma \cap \mathcal{I})$) are extended with the port p_V so as to have access to the inter-type variables and their computation function is prepended with f_b and f_a. The interactions not matched (i.e., $a \in (\gamma \setminus \mathcal{I})$) are unmodified and copied. Priorities (π) are not modified, thereby preserving the priorities on the interactions.



(a) Matching (b) Weaving

Fig. 2: Matching and Weaving a Global Aspect

Example 5. Figure 2b displays the weave on the set of interactions $\{a_1\}$ with the set of inter-type variables $V = \{v_0, v_1\}$ of the advice $\langle f_b, f_a \rangle$. A new atomic component is created B_V that has two local variables v_0 and v_1 and has its port p_V always enabled. The variables in V are attached to p_V.

- The interaction a_1 is connected to p_V so as to allow access to V on which f_b and f_a can operate.
- The computation f_b is prepended to $a_1.func$ so as to execute before and f_a is appended to $a_1.func$ so as to execute after.
- Since p_V is always enabled, the interaction a_1 will be enabled when pb_2 and pd_1 are both enabled and g_1 holds. The extension to p_V does not affect enablement.
- Once a_1 is executed if f_b or f_a write onto $p_V.vars$, they will then be received in B_V and changed accordingly.

We now define the operation that takes a global aspect, matches its pointcut expression, and weaves the result into the model.

Definition 19 (Weaving of a single global aspect). *Weaving a global aspect* $GA = \langle \mathcal{C}, V, gpc, \langle f_b, f_a \rangle \rangle$ *into a composite component* $\mathcal{C}$ *is noted* $\mathcal{C} \triangleleft_g GA$ *and yields a new composite component* $\mathcal{C}' = \mathrm{weave_g}(\mathcal{C}, \mathrm{match_g}(\mathcal{C}, gpc), V, \langle f_b, f_a \rangle)$.

Correctness of weaving. We consider $\mathcal{E}'$ (resp. $\mathcal{E}$), to be the set of reachable events in the resulting (resp. original) BIP system $\langle \mathcal{C}', q_0' \rangle$ (resp. $\langle \mathcal{C}, q_0 \rangle$) where $\mathcal{C}' = \mathcal{C} \triangleleft_g \langle \mathcal{C}, V, gpc, \langle f_b, f_a \rangle \rangle$. We begin by defining function $\mathrm{rem_g} : \mathcal{E}' \to \mathcal{E} \cup \{\epsilon\}$. Function $\mathrm{rem_g}$ removes the global advice from the event in $\mathcal{E}'$ and constructs a

similar event in $\mathcal{E}$ or the empty event ϵ if it does not match the advice.

$$\mathrm{rem_g}(\langle q_s, a, q_e\rangle) = \begin{cases} \langle q'_s, a', q'_e\rangle & \text{if } \exists f : a.func = f_b \cdot f \cdot f_a \\ \epsilon & \text{otherwise} \end{cases}$$

with:
- $a' = \langle a.ports \setminus \{p_V\}, f, a.guard\rangle$ where $a.func = \exists f : f_b \cdot f \cdot f_a$.
- q'_s and q'_e exclude the valuations of V from q_s and q_e, respectively.

The following proposition expresses the correct application of the advice on the joinpoints selected by a pointcut expression.

Proposition 2 (Weaving correctness). $\forall \langle q, a, q'\rangle \in \mathcal{E}'$:
$\exists f : a.func = \langle f_b \cdot f \cdot f_a\rangle$ *iff* $(e' \neq \epsilon \wedge e' \vDash gpc)$ *s.t.* $e' = \mathrm{rem_g}(\langle q, a, q'\rangle)$

We say that an interaction's update function satisfies an advice application if its update function ($a.func$) starts with f_b and ends with f_a (i.e., the advice's before and after update functions). This proposition states that any event's interaction satisfies an advice application iff we can construct an event e' without the advice f_b and f_a ($e' = \mathrm{rem_g}(\langle q, a, q'\rangle)$) which matches *gpc* ($e' \vDash gpc$) in the original system. Since an advice can add extra behavior like reading and writing to variables, it can cause the event to match more joinpoints, therefore it is removed before matching with *gpc*.

3.6 Overview of Local Aspects

Due to lack of space, we only give an overview of local aspects. In this view, we focus on atomic components seen as white boxes and seek to refine their behavior. An atomic component's state is studied to locate possible points where crosscutting concerns apply. Since in this view we see components as whiteboxes, we have knowledge of the full BIP system and can extract a local execution trace for a given atomic component. An atomic component has control locations, variables, and transitions labeled with ports, guards and computation functions. At this level, concerns need to be managed at the following points: port execution/enablement, guard evaluation, access and modification of state's variables (i.e., location and local variables).

The local advice defines the possible actions to be injected at a local joinpoint. Similarly to a global advice (Definition 15), a local advice executes two functions before and after the local joinpoint. The functions of a local advice may only modify the variables of the atomic component and an extra set of inter-type variables V, specific to the local aspect. Furthermore, in order to increase the expressiveness of the local advice, a local advice may change the location of the atomic component depending on a specific guard.

3.7 Weaving Multiple Aspects

When weaving multiple aspects, specific problems and extra considerations arise. Whenever a new concern is added to the joinpoint, it is possible to interfere with the existing concern at the joinpoint. This behavior is called *interference*. We manage multiple aspects by grouping them in modules (called containers). Local containers (resp. global containers) apply to local (resp. global) aspects.

Containers define an order on the aspects they encapsulate so that the weaving order is deterministic. Moreover, containers ensure that aspects share the same inter-type variables. In the case of local containers, local aspects are further required to operate on the same atomic component encouraging encapsulation, since aspects that operate on different atomic components do not interfere and cannot share inter-type variables. Weaving multiple aspects is fully described in [10].

4 Implementation

4.1 `AOP-BIP`: Aspect-Oriented Programming for BIP Systems

`AOP-BIP` is a proof-of-concept, aspect-oriented extension to BIP. `AOP-BIP` language supports both global and local aspects. Moreover, `AOP-BIP`'s command line front-end takes as input: (1) a `.bip` file that represents a BIP system written in the BIP language [22]; and (2) a list of `.abip` files that describe the aspects.`AOP-BIP` produces an output BIP model where the aspects are woven.

4.2 Example

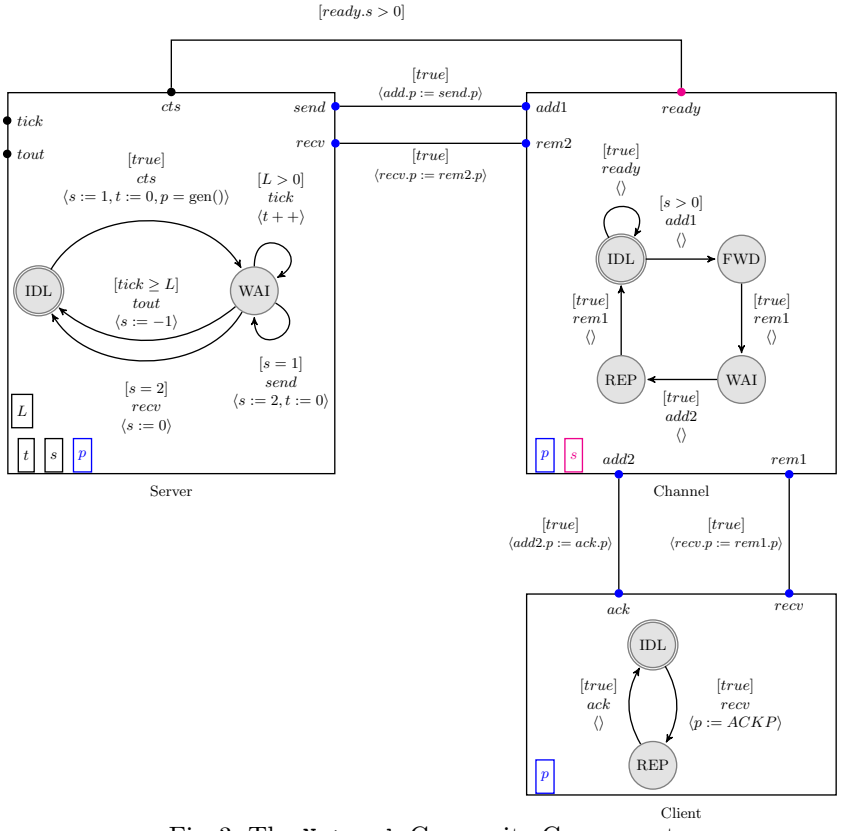


Fig. 3: The `Network` Composite Component

A network protocol is used to illustrate the handling of crosscutting concerns in BIP and is shown in Fig. 3. The `Network` composite component consists of a

```
Aspect AddHash (server)  {
  portExecute(cts)
  do {} {p = wrap(p);}
}
Aspect VerifyHash (channel) {
  data int clear = 0
  portExecute(add1)
  do {} {clear = check(p); p = unwrap(p);}
  {(IDL, clear == 0)}
}
Aspect Carol { // Man-in-the-middle
  ports(a:server.send b:channel.add1)
  readPortVars(a.r)
  do {} {b.r = pfake(a.r);}
}
```

```
[Server.cts] Clear to Send
[Server.send ] -> 886|6 (Time: 0)
[Channel.add1] <- 386|6
[Channel.rem1] -> 386
[Client.recv ] <- ACK
[Client.ack  ] -> ACK
[Channel.add2] <- ACK
[Channel.rem2] -> ACK
[Server.recv ] <- ACK (Time: 3)

[Server.cts  ] Clear to Send
[Server.send ] -> 763|3 (Time: 0)
[Channel.add1] <- 736|3
[Server.tout ] Timeout
```

Fig. 4: Authentication Aspects and Trace

`Server`, `Client` and a `Channel`. The double circles denote the start locations for each component. The `Server` waits for the *clear-to-send* signal on its `cts` port indicating that a channel is available. It then generates a packet and sends it to the channel. The channel forwards the packet to the client which acknowledges it. The channel will then send the acknowledgement back to the server.

Crosscutting concerns. The network protocol is augmented by refining its specification. The correctness of our transformation ensures that the crosscutting concerns are rigorously handled. First, logging is introduced by capturing port executions locally in all components. Second, security is added in the form of authentication. A signature (hash) is added to the packet and checked. To accomplish the above we introduce two local aspects. The various aspects along with the output are displayed in Fig. 4. The first intercepts the `Server`'s `cts` port execution and adds the signature once the server is ready to send, by modifying the packet stored in the local variable p. The second intercepts the `Channel`'s `add1` port execution, this port executes when a packet from the `Server` is sent. The advice verifies the signature (using `check(p)`) and stores the result (logical 0 or 1) in an inter-type variable `clear`. The advice also adds a reset location to `IDL` if the verification failed (`clear` $==$ `0`), preventing the `Channel` from forwarding the packet to the `Client`. The `Carol` aspect is added to modify the packet in transit to display a failed authentication. The global pointcut expression matches: (1) ports `server.send` and `channel.add1`, and (2) variable `server.send.r`. The advice in the `Carol` aspect changes the value of `channel.add1.r` after the execution of any interaction that matches the pointcut. Normally the system will execute the packet transfer by reading `a.r` and modifying `b.r`. The advice function instead will override `b.r` by generating a fake packet from `a.r` using `pfake(a.r)`. The output displays a successful and an unsuccessful attempt. The packet is represented by a number and the signature is the last digit in the number. We first notice that the first aspect added `|6` to the packet `886` and the second aspect removed it when the channel forwarded it (`Channel1.rem1`). `Carol` replaces the first packet `886` with `386`, which both have the same signature (`6`). The verification succeeds in this case unlike in the second try, when `763` is replaced by `736` since the signature of `736` is `6` but not `3`. Third, congestion avoidance is added by computing the round-trip time of the message and then waiting before

Table 1: Crosscutting Concerns in `Network`

#	Concern	Transitions	Interactions	OT	OI	OC
1	Logging	10	0	5	0	2,3
2	Authentication	2	1	2	1	1,3.4
3	Congestion	5	0	4	0	1,2
4	Fault Tolerance	0	3	0	1	2
`Network`		12	5			

sending further messages. Fourth, basic fault tolerance is introduced in the form of a failsafe mechanism. The system deadlocks and then terminates safely, after the server fails to receive a certain number of acknowledgments.

Coverage of concerns. The coverage of the concerns is shown in Table 1. Column Transitions reports the number transitions that have been modified including the number of added transitions for reset locations. Column Interactions reports the number of modified interactions. Column OT (resp. OI) reports the number of transitions (resp. interactions) that overlap with other concerns. Interfering concerns are reported in column OC. Concerns are indicated by label (1-4). We illustrate concerns that target multiple areas in the system. Without using our tool, implementing these concerns would require to edit a significant part of the system. For instance, in the case of logging, the code must be inserted in 10 transitions, of which half overlap with other concerns.

5 Related Work

AOP for CBS. Pessemier et al. [20] present a framework to deal with crosscutting concerns in CBSs. It is a symmetric approach that presents aspects as components containing the advice and additional interfaces, and are therefore integrated homogeneously within the system. Joinpoints are a combination of interfaces of different components. Thus, components are seen as black boxes. The approach has several advantages. First it explicitly models dependencies between aspects and components, and allows for their composition at an architectural level. Second, it allows the aspects to be manipulated and reconfigured at runtime. Third, it clearly defines the relationships (1) between aspects and other aspects, and (2) between aspects and the components they modify. This approach however does not consider the semantics of interactions. It targets arbitrary interface signatures, so the implementation itself must explicitly address the synchronization amongst the different components and data transfer.

Similarly, other works such as [9], [18] have been done to integrate AOP into CBSs as well. These approaches are however asymmetric and subsumed by [20]. Duclos' approach [9] defines two languages to integrate aspects. Lieberherr's approach [18] defines aspects as part of the modules they apply to, and compares the expressiveness of the approach with both AspectJ and HyperJ [21].

SAFRAN [6] differs from the above approaches by using AOP in the Fractal component model to define adaptation policies.

Formalization of aspects. None of the above approaches relies on formal models. Work to formalize aspects in programs has been undertaken by [15]. The approach specifies different categories of aspects and how they affect various

classes of properties (safety, liveness). The work has been extended by Djoko et al. [8] by expanding the categories and defining languages of aspects. The languages of aspects ensure by construction that aspects written with them fit a specific category. Additional tools for verification and analysis of aspects and their interferences have been developed and are presented in [16]. Larissa [1] is a language for handling crosscutting concerns in reactive systems modeled as the composition of Mealy automata. The matching is done by assigning monitor programs that look for a specific execution trace. Joinpoints are then associated with the input history. Advices consist of two types: `toInit` and `recovery`. The `toInit` advice places the program back in its original state. The `recovery` advice consists of restoring the program to the last recovery state it was in. A recovery state is determined by a monitor: the recovery program. The recovery states are associated with specific execution traces and are matched similarity to joinpoints. Compared to our approach, Larissa supports joinpoints based on the input history. It can also be seen as symmetric since aspects are introduced in the synchronous language used. However, the underlying model is conceived for reactive systems, and not CBSs, it does not have a clear distinction between communication and components, and thus does not distinguish between aspects related to components and communications. The communication model is based on simple input/output matching. Moreover, advices are not expressive and only consider reset/restore the state of the system. Formalizing aspects in the BRIC component model has been undertaken by [7]. BRIC formalizes component behavior and their interactions using the Communicating Sequential Processes (CSP) language. Unlike our approach, it is symmetric, aspects, pointcuts and advices are described in CSP, and woven using CSP operators. Additionally, it targets interactions and not the components themselves. It regards components as black boxes. Verification on the resulting woven system is possible in both approaches. However BIP has a strong expressive synchronization primitive [3] which is more expressive than CSP [13]. This allows more concerns to be formalized.

6 Future Work

Future work comprises five directions. The first consists in capturing more joinpoints and extending the possible behavior of advices. Possible new joinpoints include variable in interaction guard, specific values of variables. Advices can be extended to modify guards on matching transitions and interactions. The second consists in applying CBS methods to define advices and aspect composition. This helps integrate AOP in BIP symmetrically, where aspects are implemented as components and interactions within the existing system. This would allow to enable or disable aspects in the system, and specify more complex advices (i.e., advices as components instead of update functions and extra transitions). The third consists in elaborating new ways to compose aspects by finding new criteria to order them. Aspects can be re-ordered in a container based on their pointcut expressions, by grouping those which affect the same transitions or interactions, and whether or not they modify the existing variables (read/write aspects). Additionally, the language could be extended to allow the explicit def-

inition of precedence rules. The fourth consists in implementing model-to-model transformations using Domain Specific Languages inspired by ATL [14] targeting the BIP model and comparing their expressiveness with our approach. The fifth is to integrate existing crosscutting concerns (such as monitoring [11]) with `AOP-BIP`, and in particular existing work such as [12, **?**].

Acknowledgement. The work reported in this article is in the context of the COST Action ARVI IC1402, supported by COST (European Cooperation in Science and Technology). Moreover, the authors acknowledge the support of the University Research Board (URB) at American University of Beirut.

References

1. Altisen, K., Maraninchi, F., Stauch, D.: Aspect-oriented programming for reactive systems: Larissa, a proposal in the synchronous framework. Sci. Comput. Program. 63(3), 297–320 (2006)
2. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the BIP framework. IEEE Software 28(3), 41–48 (2011)
3. Bliudze, S., Sifakis, J.: A notion of glue expressiveness for component-based systems. In: Proc. of the 19th Int. Conf. on Concurrency Theory. pp. 508–522 (2008)
4. Bozga, M., Jaber, M., Sifakis, J.: Source-to-source architecture transformation for performance optimization in BIP. IEEE Trans. Industrial Informatics 6(4), 708–718 (2010)
5. Czarnecki, K., Eisenecker, U.W., Steyaert, P.: Beyond objects: Generative programming. In: The 23rd Int. Conf. On Software Engineering. pp. 5–14 (1997)
6. David, P., Ledoux, T.: An aspect-oriented approach for developing self-adaptive Fractal components. In: Proc. of 5th Int. Symposium on Software Composition. LNCS, vol. 4089, pp. 82–97 (2006)
7. Dihego, J., Sampaio, A.: Aspect-oriented development of trustworthy component-based systems. In: Proc. of Theoretical Aspects of Computing. LNCS, vol. 9399, pp. 425–444 (2015)
8. Djoko, S.D., Douence, R., Fradet, P.: Aspects preserving properties. Sci. Comput. Program. 77(3), 393–422 (2012)
9. Duclos, F., Estublier, J., Morat, P.: Describing and using non functional aspects in component based applications. In: AOSD. pp. 65–75 (2002)
10. El-Hokayem, A., Falcone, Y., Jaber, M.: http://ujf-aub.bitbucket.org/aop-bip/
11. Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. In: Engineering Dependable Software Systems, vol. 34, pp. 141–175. IOS Press (2013)
12. Falcone, Y., Jaber, M.: Fully automated runtime enforcement of component-based systems with formal and sound recovery. International Journal on Software Tools for Technology Transfer pp. 1–25 (2016)
13. Falcone, Y., Jaber, M., Nguyen, T., Bozga, M., Bensalem, S.: Runtime verification of component-based systems in the BIP framework with formally-proved sound and complete instrumentation. Software and System Modeling 14(1), 173–199 (2015)
14. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
15. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comput. Program. 72(1-2), 31–39 (2008)
16. Katz, S.: Aspect categories and classes of temporal properties. In: Transactions on aspect-oriented software development I, pp. 106–134. Springer (2006)
17. Katz, S., Faitelson, D.: The common aspect proof environment. STTT 14(1), 41–52 (2012)

18. Lieberherr, K.J., Holland, I.M.: Formulations and benefits of the law of demeter. SIGPLAN Notices 24(3), 67–78 (1989)
19. Lieberherr, K.J., Lorenz, D.H., Ovlinger, J.: Aspectual collaborations: Combining modules and aspects. Comput. J. 46(5), 542–565 (2003)
20. Noureddine, M., Jaber, M., Bliudze, S., Zaraket, F.A.: Reduction and abstraction techniques for BIP. In: Proc. of the 11th Int. Symposium on Formal Aspects of Component Software. LNCS, vol. 8997, pp. 288–305 (2014)
21. Pessemier, N., Seinturier, L., Duchien, L., Coupaye, T.: A component-based and aspect-oriented model for software evolution. IJCAT 31(1/2), 94–105 (2008)
22. Tarr, P., Ossher, H.: Hyper/J: Multi-dimensional separation of concerns for Java. In: Proc. of the 23rd Int. Conf. on Software Engineering. pp. 729–730 (2001)
23. Verimag: BIP Tools, http://www-verimag.imag.fr/BIP-Tools,93.html

A Proofs

We first prove that the global joinpoints can be selected syntactically (Proposition 1), and then prove the correctness of advice application (Proposition 2).

Proof (Proof of Proposition 1). We consider a global event $\langle q, a, q' \rangle \in \mathcal{E}$ and a global pointcut expression $\langle p, v_r, v_w \rangle$. The proof follows from the definition of $\text{match}_\text{g}(\mathcal{C}, gpc)$ which selects all interactions matching the criteria that should be matched by $(E_i = \langle q, a, q' \rangle \vDash gpc)$.

$$
\begin{aligned}
(\langle q, a, q' \rangle \vDash \langle p, v_r, v_w \rangle) \text{ iff } & p \subseteq a.ports \\
& \wedge\, v_r \subseteq \text{readvar}(a.func) \\
& \wedge\, v_w \subseteq \text{writevar}(a.func) && \text{(Definition 14)} \\
\text{iff } & a \in \text{match}_\text{g}(\gamma, gpc) && \text{(Def. of match}_\text{g}\text{)}
\end{aligned}
$$

Proof (Proof of Proposition 2). We assume that f_b and f_a can be uniquely determined and are not empty. It is possible to add an assignment statement at the start and end of each f_b and f_a which has no effect and is not present in the original system and acts as a marker (example: $x = x + NUM - NUM$, with NUM being a unique number not found in any other assignment). We consider an event $\langle q_s, a, q_e \rangle \in \mathcal{E}'$, and $\text{rem}_\text{g}(\langle q_s, a, q_e \rangle) = \langle q'_s, a', q'_e \rangle$ the constructed event without the advice.

$$
\begin{aligned}
\exists f : a.func = \langle f_b \cdot f \cdot f_a \rangle \text{ iff } & \exists a' \in \gamma : \text{m}(a') = a \wedge a' \in \mathcal{I}) && \text{(Definition 18, m())} \\
\text{iff } & a' \in \text{match}_\text{g}(\mathcal{C}, gpc) && \text{(Definition 19)} \\
\text{iff } & (\langle q'_s, a', q'_e \rangle \vDash gpc) && \text{(Proposition 1)} \\
\text{iff } & \text{rem}_\text{g}(\langle q_s, a, q_e \rangle) \vDash gpc)
\end{aligned}
$$

An event interaction's update function $a.func$ starts with f_b and ends with f_a, according to the definition of m() in Definition 18 iff it is the result of weaving the advice on it from an interaction a' ($\exists a' \in \gamma : \text{m}(a') = a \wedge a' \in \mathcal{I}$). The interaction a' is in $\mathcal{I}$ iff it was selected by the local pointcut expression ($a' \in \text{match}_\text{g}(\mathcal{C}, gpc)$). According to Proposition 1, any event with interaction $a' \in \text{match}_\text{g}(\mathcal{C}, gpc)$ is a joinpoint, specifically $\langle q'_s, a', q'_e \rangle = \text{rem}_\text{g}(\langle q_s, a, q_e \rangle)$.