**Introdução**

O paradigma de orientação a aspectos envolve duas etapas de trabalho. A primeira é a decomposição do sistema em partes não entrelaçadas e não espalhadas. A segunda envolve juntar essas partes novamente de forma significativa para se obter o sistema desejado. O processo de juntar as partes se chama composição. Há três questões a serem definidas em qualquer linguagem orientada a aspectos para se fazer a composição: a correspondência, a semântica composicional, e o tempo de ligação.

* **Correspondência** - A forma de correspondência da linguagem é o modo com o qual se descreve quais entidades serão compostas entre si. A correspondência pode ser implícita (determinada por regras da linguagem) ou explícita (descrita pelo programador).
* **Semântica Composicional** - é o que deve acontecer com os elementos que correspondem. Em geral, linguagens de POA modificam a semântica das chamadas a métodos:
  + Em linguagens procedurais, chamar a função F implica em executar a função F.
  + Em linguagens orientadas a objetos, chamar o método M implica em executar algum método M em uma das subclasses que definem M.
  + Em linguagens orientadas a aspectos, chamar o método M pode ter diversas consequências:
    - M é executado, ou
    - N (algum outro método) é executado, ou
    - M+N são executados, em alguma ordem definida

A linguagem pode definir diversas semânticas diferentes que em geral são escolhidas pelo programador.

* **Tempo de Ligação** - diz respeito ao momento em que a correspondência passa a surtir efeito; pode ser estático (em tempo de compilação) ou dinâmico (em tempo de execução).

A ferramenta ou compilador que faz a composição dos elementos em POA é chamado weaver (tecelão), pois "tece" os vários fragmentos de programa em um programa único.

A forma de composição das partes é o que realmente distingue linguagens orientadas a aspectos de outras linguagens. Em linguagens procedurais ou orientadas a objetos, a composição é feita através de chamadas de procedimentos ou métodos. Ou seja, uma parte (por exemplo, uma classe) usa a funcionalidade de outra chamando um método.

Em POA, não há chamadas explícitas de métodos entre partes. Ao invés disso, especifica-se, em uma parte separada, como uma parte deve reagir a eventos que acontecem em outra parte. Essa estratégia reduz o acoplamento entre as partes, pois as partes não se acessam diretamente.

**Vantagens**

* **Menos responsabilidades em cada parte** - Como interesses entre cortantes são separados em seus próprios módulos, as partes do programa que lidam com a lógica de negócios não ficam poluídas com código que lida com interesses periféricos;
* **Melhor modularização** - Como os módulos em POA não se chamam diretamente, há uma redução no nível de acoplamento;
* **Evolução facilitada** - Novos aspectos podem ser acrescentados facilmente sem necessidade de alterar o código existente;
* **Mais possibilidades de reutilização** - Como o código não mistura interesses, aumentam-se as possibilidades de se reutilizar módulos em sistemas diferentes.

**Desvantagens**

* **É difícil seguir a lógica de programas orientados a aspectos** - Como um módulo não chama outro diretamente, é difícil inferir o comportamento do sistema como um todo avaliando-se módulos individuais.
* **A POA não resolve nenhum problema novo** -A POA não tenta resolver problemas não solucionados. Há outras soluções para os problemas de espalhamento e entrelaçamento que não envolvem a criação de novas tecnologias.