UNIVERSIDADE ABERTA
www.uab.pt

UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

Mestrado em Engenharia Informática e Tecnologia Web

Arquitetura e Padrões de Software

(22304)

Ano letivo 2025/2026

Aplicação do Padrão de Criação Factory Method no ProPlan

Síntese

ProPlan é um módulo *Activity Provider* que apresenta ao aluno uma série de desafios de gestão de projeto simulados, baseados em cenários realistas.

André Sousa

1300012@estudante.uab.pt

Aplicação do Padrão de Criação Factory Method no ProPlan – Activity Provider

1. Introdução

A evolução das arquiteturas de software para modelos modulares, distribuídos e facilmente extensíveis tem reforçado o papel dos padrões de software como instrumentos de comunicação, formalização e construção de sistemas robustos (Gamma et al., 2000). No ecossistema da Inven!RA, um Activity Provider deve disponibilizar serviços REST que obedeçam a requisitos de consistência, autonomia, testabilidade e extensibilidade.

Entre estes serviços, o endpoint analytics_url assume particular importância, dado que fornece dados analíticos sobre a execução de atividades por parte dos estudantes. A adoção do padrão Factory Method no seu desenho permite separar responsabilidades, isolar pontos de variação e cumprir princípios como OCP (Open/Closed Principle) e DIP (Dependency Inversion Principle).

2. Identificação do problema

Na versão inicial do projeto, o *endpoint*:

POST /analytics-proplan

construía diretamente a resposta JSON, lendo o ficheiro analytics_url.json e gerando valores analíticos no próprio controlador Flask. Este desenho implicava três limitações:

1. **Acoplamento elevado:**
   O controlador conhecia a origem, o formato e a lógica de construção dos dados.
2. **Dificuldade de evolução:**
   Alterar a fonte de dados (JSON → base de dados → API externa) exigiria modificar o endpoint.
3. **Ausência de ponto de variação isolado:**
   Não havia abstração que permitisse trocar o mecanismo de obtenção de analytics.

Estas limitações enquadram-se em violações clássicas mitigadas através da aplicação do **Factory Method** (Gamma et al., 2000).

3. Aplicação do padrão *Factory Method*

Para eliminar o acoplamento e preparar o sistema para evolução futura, foi introduzida uma classe responsável pela criação do repositório de analytics – a **RepositoryFactory**.

A estrutura final inclui:

- a interface abstrata **AnalyticsRepository**;

- a implementação concreta **JsonAnalyticsRepository**;
- a fábrica **RepositoryFactory** com o método create_analytics_repository().

O controlador Flask deixa de instanciar diretamente o repositório, passando a depender apenas da abstração.

Exemplo:

```
repo = RepositoryFactory.create_analytics_repository()
analytics = repo.get_analytics(activity_id)
```

A fábrica devolve atualmente um JsonAnalyticsRepository(), mas poderá futuramente instanciar:

- SqlAnalyticsRepository()
- ApiBasedAnalyticsRepository()
- MockAnalyticsRepository()

sem alterações ao endpoint — cumprindo o princípio de extensibilidade.

4. Arquitetura proposta

A aplicação do padrão implicou a criação de três elementos estruturantes:

4.1. Interface – *AnalyticsRepository*

Define o contrato obrigatório para qualquer implementação de repositório:

```
class AnalyticsRepository(ABC):
  @abstractmethod
  def get_analytics(self, activity_id: str) -> list[dict]:
    pass
```

4.2. Implementação concreta – JsonAnalyticsRepository

Lê o esquema *analytics_url.json* e devolve dados analíticos simulados.

```
class JsonAnalyticsRepository(AnalyticsRepository):
  def get_analytics(self, activity_id: str) -> list[dict]:
    ...
```

4.3. Fábrica – *RepositoryFactory*

Responsável por decidir qual repositório instanciar.

```
class RepositoryFactory:
    @staticmethod
    def create_analytics_repository() -> AnalyticsRepository:
        return JsonAnalyticsRepository()
```

4.4. Controlador Flask (refatorado)

```
@app.post("/analytics-proplan")
def analytics_proplan():
    data = request.get_json(silent=True) or {}
    activity_id = data.get("activityID")
    repo = RepositoryFactory.create_analytics_repository()
    analytics = repo.get_analytics(activity_id)
    return jsonify(analytics)
```

4.5. Inclusão do serviço analytics_list_url (Analytics Contract)

A arquitetura Inven!RA distingue dois serviços:

- **analytics_list_url** – *contrato dos analytics* (lista de métricas suportadas)
- **analytics_url** – *valores dos analytics* (dados específicos da instância)

Assim, o ProPlan expõe também:

GET /analytics-list-proplan

que fornece o **contrato dos analytics**, representado no diagrama através da interface **IAnalyticsContract**, provida pelo ProPlan e requerida pela Inven!RA.

5. Diagrama de Componentes (segundo a norma UML 2)

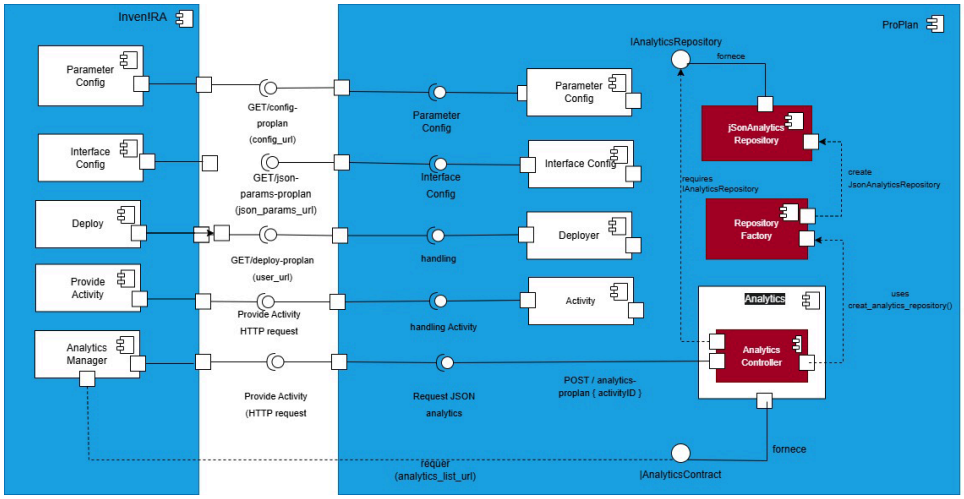

Figura 1 - Diagrama Componentes

Características verificadas, de acordo com a norma:

- Componentes representados com símbolo UML2.
- Interface *IAnalyticsRepository* modelada como elemento UML separado.
- Interface providenciada por *JsonAnalyticsRepository* (*lollipop*).
- Interface requerida pelo porto *repoPort* do controlador *Flask* (*socket*).
- Componentes

6. Diagrama de Sequência (só o padrão de criação)

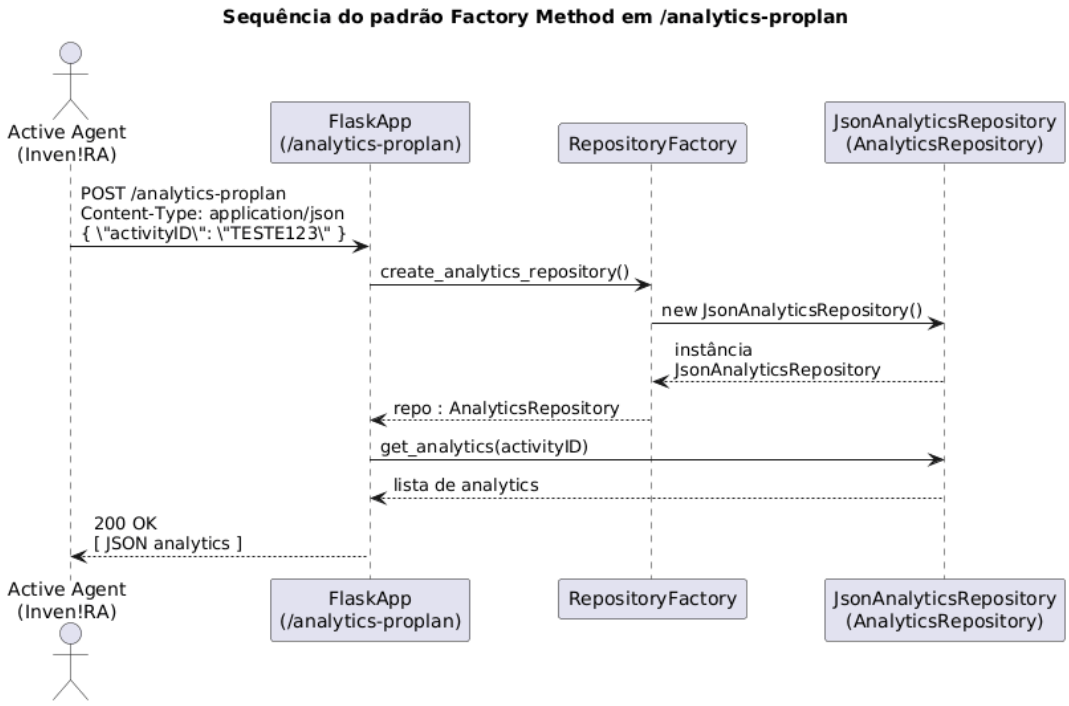

Figura 2 - Diagrama Sequência

O diagrama representa apenas:

- o pedido da Plataforma Inven!RA,
- a invocação da fábrica,
- a criação de *JsonAnalyticsRepository*,
- a chamada *get_analytics*(*activityID*),
- a resposta 200 OK.

O diagrama não inclui nenhum outro endpoint, cumprindo rigorosamente a orientação do docente.

7. Conclusão

A introdução do padrão **Factory Method** no *Activity Provider ProPlan* permitiu alcançar um desenho mais robusto, extensível e conforme às boas práticas de engenharia de software. O controlador passa a depender apenas de abstrações, reduzindo acoplamento e facilitando a evolução futura do sistema. A representação estrutural e comportamental (através dos diagramas de componentes e de sequência) demonstra formalmente a aplicação do padrão, alinhando-se com as orientações da UC e com referências clássicas da engenharia de software.

Referências (ABNT)

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Padrões de projeto: soluções reutilizáveis de software orientado a objetos*. Porto Alegre: Bookman, 2000.

MORGADO, Leonel; CASSOLA, Fernando. *Activity Providers na Inven!RA*. Lisboa: Universidade Aberta, 2022.

FAKHROUTDINOV, Kirill. *UML Component Diagrams: Reference*. uml-diagrams.org, 2025.

SOUSA, André. *ProPlan – Activity Provider*. GitHub, 2025. Disponível em: https://github.com/AndreMacielSousa/proplan-activity-provider. Acesso em: 26 nov. 2025.