Tiago Delgado Dias
Software Architect

Empresa: Brisa, S.A.
E-mail: tdias@ptsoft.net

DESIGN PATTERNS & MDSOC: CASOS DE USO EM SOFTWARE ITS

A plataforma *iBrisa* é uma arquitectura aplicacional utilizada na área de operações da Brisa. É composta por várias aplicações encarregues de áreas de negócio específicas e entre as quais existem pontos de integração, sendo a maioria através de serviços web.
O *Traffic Atlas* é uma das aplicações nucleares da plataforma *iBrisa*. Oferece uma interface web para a representação da infra-estrutura rodoviária, equipamentos de telemática, incidências e meios. Suporta ainda a operação, gestão e manutenção de equipamentos de telemática e a gestão da infra-estrutura. Através da integração com sistemas específicos oferece uma interface unificada para processos desde a gestão até à manutenção.
Ao longo do desenvolvimento da plataforma *iBrisa* e em particular do *Traffic Atlas* foram utilizadas soluções comuns a problemas recorrentes. A abordagem destas soluções é facilmente padronizável, encontrando-se estes padrões catalogados como *design patterns*. A utilização de alguns destes padrões trouxe no caso do *iBrisa* várias vantagens para o negócio, nomeadamente facilitando a evolução da plataforma e diminuindo tempos de desenvolvimento, em particular de requisitos transversais (que abrangem vários módulos). Iremos assim focar estes padrões e as suas mais valias.

Component pattern [1]

Este padrão promove a modularização do *software* por camadas de funcionalidade, será provavelmente o padrão mais utilizado no *software* contemporâneo. Os componentes de camadas inferiores (acesso a dados, serviços) oferecem as suas funcionalidades às camadas superiores. Cada aplicação da plataforma *iBrisa* encontra-se decomposta em componentes, tipicamente: camada de dados, de serviços e interface. Para além da separação das camadas em componentes utilizam-se ainda no *iBrisa* componentes de interface (*web controls*) específicos para permitir a reutilização em diversos cenários. Sendo a autenticação uma funcionalidade transversal e necessária num grande conjunto de páginas das interfaces criou-se um *web control* que modulariza esta funcionalidade, estando presente em páginas de inúmeras componentes (mesmo aplicações). Através deste padrão foi possível utilizar uma componente única de autenticação cuja

evolução é, até certo ponto, independente das páginas que a utilizam.

***Provider/Adapter pattern* [2]**
Este padrão funciona como mediador entre interfaces distintas. É de extrema utilidade sempre que exista um conjunto de serviços/equipamentos que ofereçam a sua funcionalidade através de interfaces proprietárias. Pode ser usado com o objectivo de oferecer uma interface normalizada, abstraindo as interfaces proprietárias, ou de forma a manter uma aplicação integradora tão simples quando possível. Este último é o caso do *Traffic Atlas*, em que se normalizou a interface de interacção com vários tipos de equipamentos e serviços e depois se criaram diferentes *providers* para cada interface distinta que era necessário integrar. Novas interfaces proprietárias são integradas através do desenvolvimento de um novo *provider* sem ser necessária qualquer alteração ao *Traffic Atlas*. Para além da aplicação na interacção com equipamentos de telemática existem também *providers* para a gestão de canais de comunicação e para os canais de distribuição de notícias.

***Observer pattern* [2]**
Através de um mecanismo de eventos este padrão permite definir várias acções de resposta a um evento despoletado. Pode ser utilizado para suportar dependências não explícitas entre objectos, contudo é utilizado no *Traffic Atlas* para desacoplar acções transversais do código operacional. É utilizado para registar várias acções de tratamento de alarmes, edição e operação de equipamentos. No caso dos alarmes, uma acções regista o alarme na base de dados, outra actualiza o estado do equipamento e outra gera uma notificação para operadores humanos (em casos mais particulares).

***Hook pattern* [3]**
O padrão do observador não permite que as acções definidas influenciem directamente a origem dos eventos, existem contudo casos em que isso é desejável, mantendo a separação oferecida pelo padrão do observador. Este padrão estende um mecanismo de eventos de forma a que as acções registadas para cada evento definam em conjunto o valor do tipo de retorno do evento. No *Traffic Atlas* este padrão é utilizado quando um utilizador pretende terminar ou transferir a sua sessão, existindo inúmeras condicionantes que podem não permitir a execução da operação. A avaliação destas condições é efectuada em vários módulos independentes que registam o *hook* de alteração de sessão e compõem respostas parciais que depois é combinada para determinar se é possível a alteração da sessão.

A modularização e separação são objectivos comuns aos padrões apresentados. As componentes separam diferentes módulos e camadas e definem os seus pontos de integração através de interfaces. Os *providers* separam a integração com interfaces heterógeneas para o mesmo tipo de serviço, permitindo que uma componente ofereça esse serviço de uma forma unificada. Os observadores e *hooks* separam o tratamento que é dado a eventos específicos do contexto em que o evento é detectado. Estes padrões são inteiramente suportados pela programação orientada a objectos (OO).
Existem contudo limitações ao uso destes padrões; por exemplo existem funcionalidades dispersas por várias componentes, em particular

funcionalidades relacionadas com requisitos não funcionais como segurança e autenticação. Existem componentes que acumulam várias funcionalidades e cuja separação dá origem a novos problemas. Já o padrão do observador implica que se acrescentem lançamentos de eventos no código operacional.
Estes problemas são endereçados por novas abordagens de programação que pretendem estender a programação orientada aos objectos. Uma destas abordagens é a separação multidimensional de conceitos (do inglês multidimensional separation of concerns - MDSoC). Esta abordagem promove a separação dos objectos no espaço multidimensional das funcionalidades. São depois definidos mecanismos de composição que juntam as componentes dos objectos nas várias dimensões de forma a gerar código passível de compilação numa plataforma típica (OO).
Através da criação de um protótipo desta abordagem [4] e da sua aplicação a pequenos exemplos da área de ITS foi possível separar totalmente as funcionlidades que não se tinham conseguido separar através da utilização de padrões. Com a aplicação desta abordagem ao *Traffic Atlas* seria possível facilitar a análise do código (para o programador e *code-reviewer)*, agilizar a localização de componentes afectadas por requisitos de alteração de funcionalidades, minimizar os conflitos obtidos no controlo de versões e mapear requisitos funcionais e não funcionais até ao código.
Foi ainda possível aproveitar a separação multidimensional para gerar versões com diferentes combinações de funcionalidades dos mesmos exemplos. Esta capacidade é essencial numa linha de produto, como o *Traffic Atlas*, para a geração de versões com diferentes combinações de funcionalidades (por exemplo para instalação em diferentes cenários).

References

[1] B. Appleton, *"Component Design Patterns*". http://c2.com/ppr/wiki/ComponentDesignPatterns/ComponentDesignPatterns.html, 2005.

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides*, "Design Patterns: Elements of Reusable Object-Oriented Software"*, pgs. 139 and 293. Addison-Wesley, 1995.

[3] S. Black, *"Design Pattern: Hook Operations*". http://www.stevenblack.com/PTN-HookOperations.asp, July, 2000.

[4] T. Dias, A. Moreira, *"Hyper/Net: MDSoC Support for .NET*". DSOA 2006 - JISBD, Sitges, Barcelona, October, 2006.