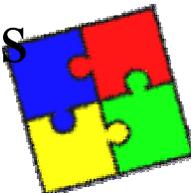
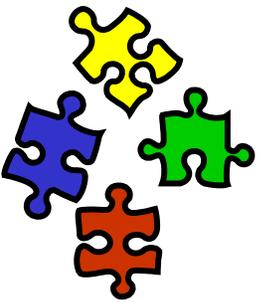


Estilo: BlackBoard

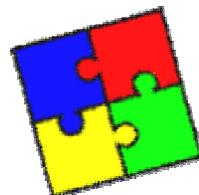
- Útil para problemas no qual não há uma solução determinística
- Uma coleção de programas independentes que trabalham cooperativamente em uma estrutura de dados comum (blackboard)
 - Vários subsistemas especializados agregam seu conhecimento para conseguir uma possível solução aproximada para o problema
 - Os subsistemas especializados são independentes uns dos outros
- BlackBoard = repositório de dados compartilhados

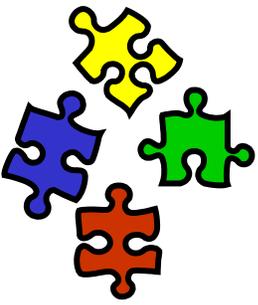




Estilo: BlackBoard

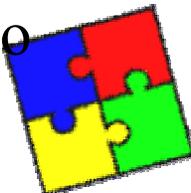
- Justificativa da denominação BlackBoard
 - Herda da idéia de pessoas que trabalham juntas na frente de um quadro (blackboard) para resolver uma tarefa.

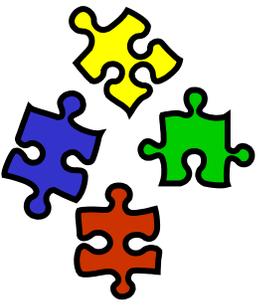




Estilo: BlackBoard

- Componentes:
 - **Origens do Conhecimento:** elementos independentes que resolvem aspectos específicos do problema. Juntos modelam o domínio do problema. Nenhum pode resolver a tarefa do sistema sozinho.
 - **Blackboard:** elemento central de armazenamento. Armazena dados para resolver o problema que são modificados pelos componentes origens do conhecimento
 - **Controle:** executa um loop que monitora o estado do blackboard e decide qual a próxima ação que tipicamente é o escalonamento de algum elemento *origem do conhecimento*

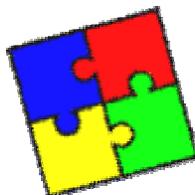
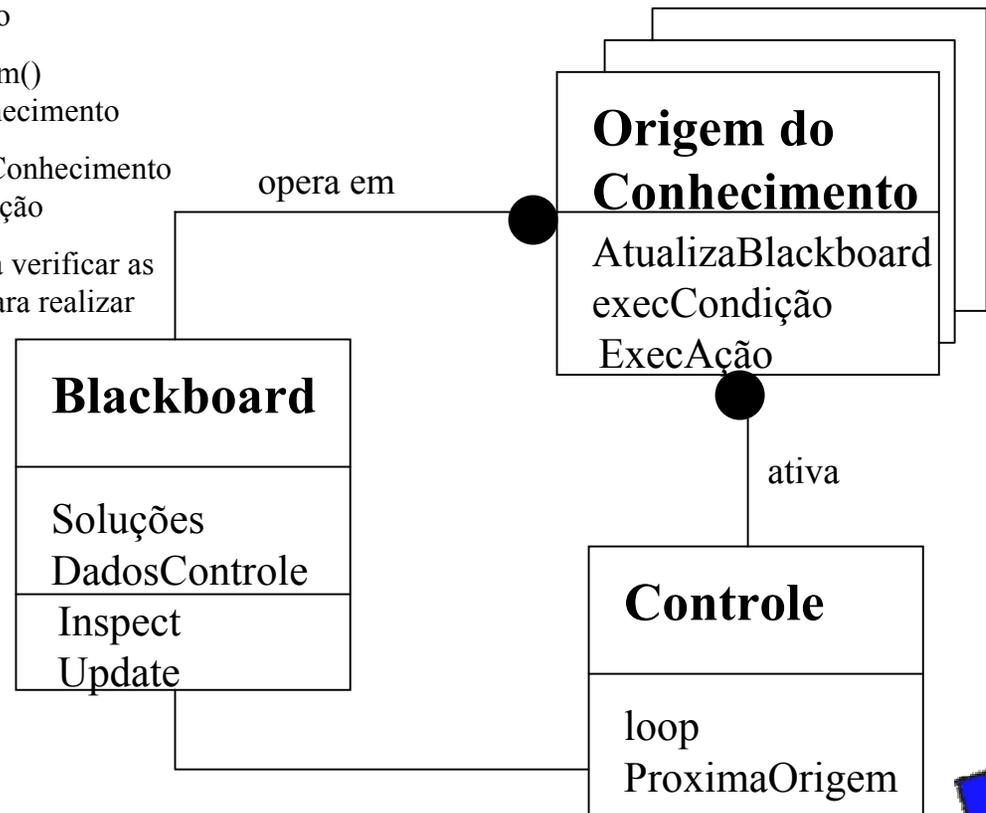


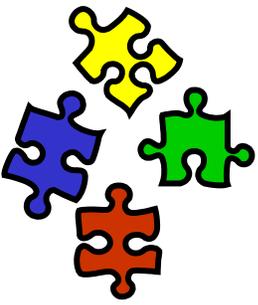


Estilo: BlackBoard

Comportamento:

- O main loop do componente Controle é iniciado
- Controle invoca o procedimento ProximaOrigem() para seleccionar o componente Origem do Conhecimento
- ProximaOrigem() determina quais Origem do Conhecimento são potenciais candidatos para encontrar a solução
- Origem do Conhecimento invoca Inspect() para verificar as soluções correntes no blackboard e Update() para realizar mudanças nos dados do blackboard

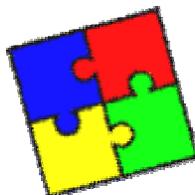


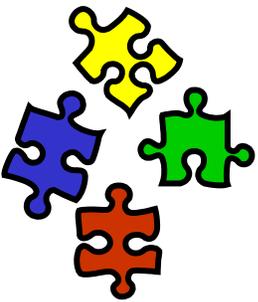


Estilo: BlackBoard

Class	Colaboradores	Class	Colaboradores
BlackBoard	-	Origem do Conhecimento	BlackBoard
Responsabilidade Gerencia dados centrais		Responsabilidade Avalia sua aplicabilidade Computa um resultado Atualiza o BlackBoard	

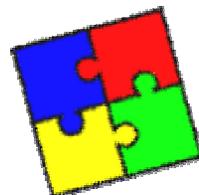
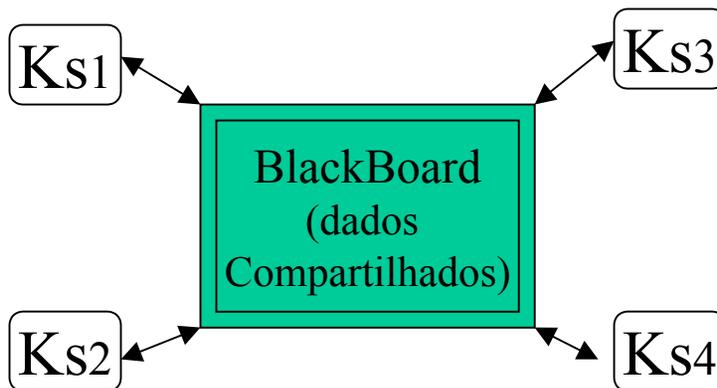
Class	Colaboradores
Controle	Blackboard Origem do Conhecimento
Responsabilidade Monitora Blackboards Escalona as ativações da origem do conhecimento	

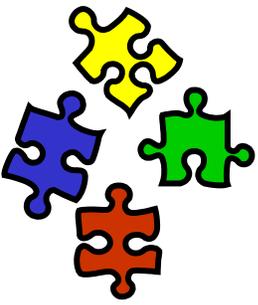




Estilo: BlackBoard

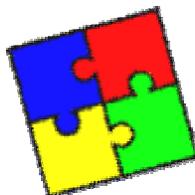
- Tem sido utilizado em aplicações que necessitam de complexas interpretações de reconhecimento de sinais como reconhecimento de fala, de padrões, de imagem

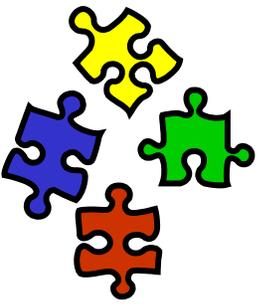




Estilo: BlackBoard

- Variações do Estilo:
 - *Repositório*: a estrutura de dados central é um repositório. Não especifica um controle. Um Banco de Dados tradicional pode ser considerado um repositório.

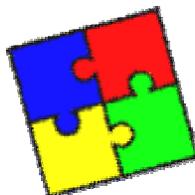


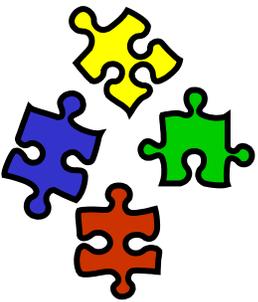


Estilo: BlackBoard

Implementação

- Defina o problema:
 - Especifique o domínio do problema e os campos de conhecimento necessários para encontrar uma solução
 - Defina a entrada e saída desejável do sistema
- Defina o espaço de solução para o problema
- Divida o processo da solução em passos:
 - Especifique os tipos de conhecimento que podem ser usados para excluir partes do espaço de solução
- Divida o conhecimento em origens de conhecimento especializadas

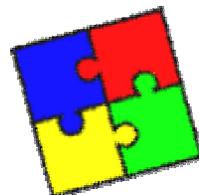


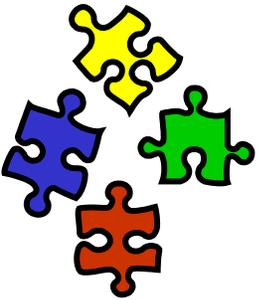


Estilo: BlackBoard

Implementação – cont.

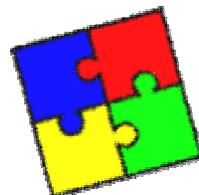
- Especifique o controle do sistema
 - Determine quais origens do conhecimento tem permissão para realizar mudanças no blackboard
 - Associe as categorias de mudanças no blackboard com um conjunto de possíveis origens do conhecimento
- Implemente as origens do conhecimento
 - Divida-os em partes de condição e partes de ação

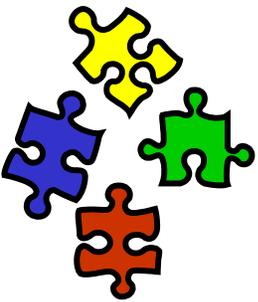




Estilo: Broker

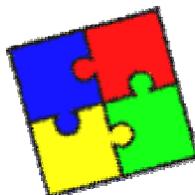
- Contexto:
 - Construção de um sistema complexo como um conjunto de componentes distribuídos que necessitam comunicar-se. São necessários serviços para adicionar, remover, trocar, ativar e localizar componentes.
- No estilo Broker componentes devem ser capazes de acessar serviços oferecidos por outros através de invocações remotas e transparentes realizadas via um componente *Broker* que desacopla servidores de clientes.

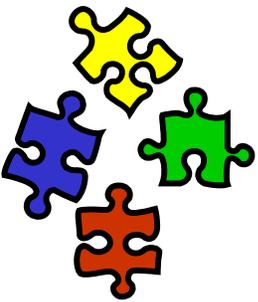




Estilo: Broker

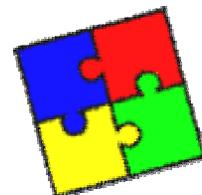
- Usado para estruturar sistemas distribuídos com componentes desacoplados que interagem por invocações remotas ao serviço
- O componente *Broker* é responsável por coordenar a comunicação entre os componentes distribuídos

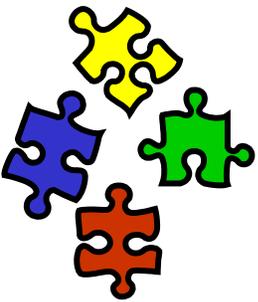




Estilo: Broker

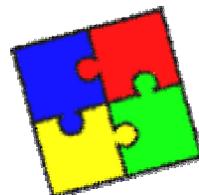
Class Broker	Colaboradores Clientes Servidores Pontes	Class Servidor	Colaboradores Broker
Responsabilidade Registrar servidores Localizar servidores Transferir mensagens Interoperar com outros brokers através de Pontes		Responsabilidade Implementa serviços Registra-se com o Broker Envia respostas ao cliente	
		Class Cliente	Colaboradores Broker
		Responsabilidade Enviar solicitações a servidores	

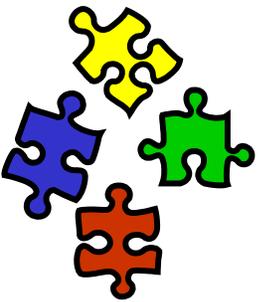




Estilo: Broker

Class Ponte	Colaboradores Broker Ponte
Responsabilidade Encapsula funcionalidade específica da rede Realiza mediação entre o broker local, a ponte e o broker remoto	

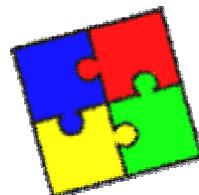


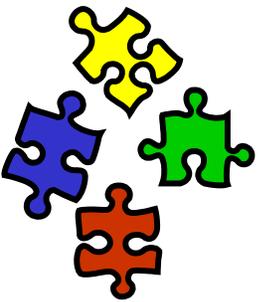


Estilo: Broker

Implementação

- Defina um modelo de objetos ou use um modelo existente
- Defina qual tipo de interoperabilidade entre componentes o sistema deve oferecer
 - Em geral usa-se uma Linguagem de Definição de Interfaces
- Especifique a API que o componente Broker oferece para interagir com clientes e servidores

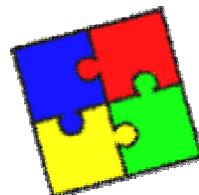


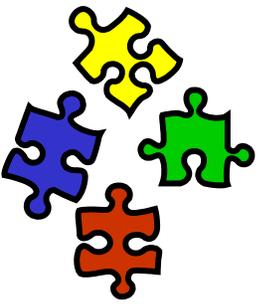


Estilo: Broker

Implementação – cont.

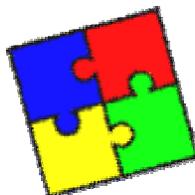
- Projete o componente Broker
 - Especifique detalhes da interação do cliente com o servidor
 - Especifique o comportamento do sistema em caso de falhas

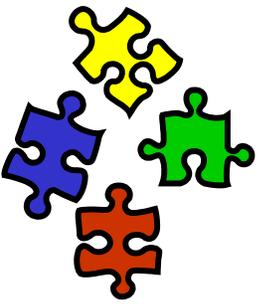




Estilo: Broker

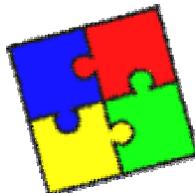
- Benefícios:
 - **Transparência de Localização:** como o broker é responsável por localizar um servidor usando um único identificador, clientes não precisam saber onde os servidores estão.
 - **Mudança e Extensibilidade dos Componentes:** se os servidores mudam a implementação mas mantêm a interface, não há impacto para os clientes.

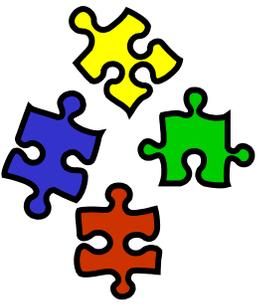




Estilo: Broker

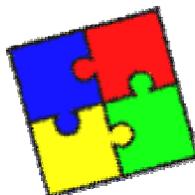
- Benefícios (cont.):
 - **Portabilidade do sistema Broker:** o broker esconde o sistema operacional e detalhes da rede usando camadas de indireção. Quando portabilidade é necessária é suficiente, em alguns casos, portar o broker. Para isto, é interessante que o broker seja estruturado usando um estilo em camadas onde as camadas mais baixas escondam detalhes específicos do sistema.
 - **Reusabilidade:** ao construir novas aplicações pode-se reusar componentes

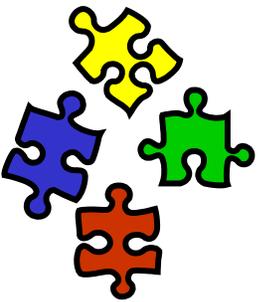




Estilo: Broker

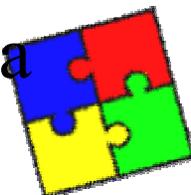
- Variações do Estilo:
 - *Broker de Comunicação Direta*: relaxa a restrição que clientes podem direcionar solicitações através do broker local. Nesta variante clientes podem comunicar-se diretamente com servidores. O broker informa ao cliente qual o canal de comunicação que o servidor oferece
 - *Sistema Trader*: uma solicitação do cliente é usualmente encaminhada a exatamente um único servidor. Em um sistema Trader o broker apenas conhece quais servidores podem oferecer o serviço e encaminha a solicitação ao servidor apropriado

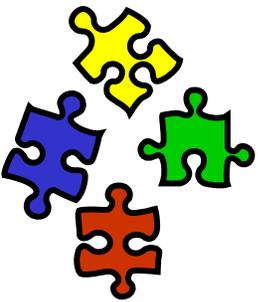




Estilo: Reflexão

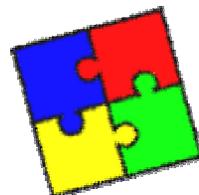
- Contexto:
 - A construção de sistemas que suportam sua própria modificação
 - Necessidade de tornar aspectos selecionados da sua estrutura e comportamento acessíveis para adaptação
- Provê um mecanismo para mudar dinamicamente a estrutura e comportamento dos sistemas
- Permite a modificação de aspectos fundamentais: estruturas de tipos e mecanismos de chamadas de funções
- Sinônimos: Implementação Aberta, Arquitetura Meta Nível

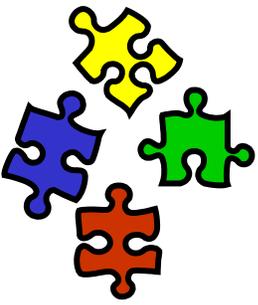




Estilo: Reflexão

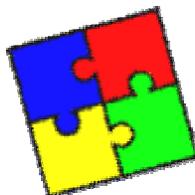
- Sistemas de software evoluem com o tempo. Eles devem estar abertos a modificações em resposta a mudanças nas tecnologias e requisitos. É necessário especificar uma arquitetura que seja aberta a modificação e extensão.

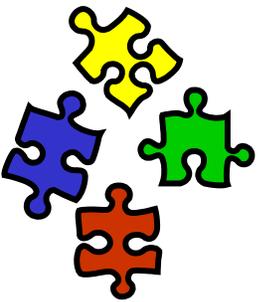




Estilo: Reflexão

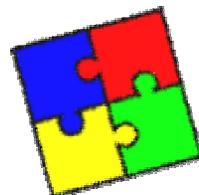
- Divide uma aplicação em duas partes:
 - Um **meta nível** que oferece informações sobre sua estrutura e comportamento e consiste dos *metaobjetos* que tornam a aplicação *self-aware*. Metaobjetos encapsulam e representam informações sobre a aplicação: estruturas de tipos, algoritmos.
 - Um **nível base** que inclui a lógica da aplicação. Sua implementação usa os metaobjetos. Por exemplo, componentes do nível base podem apenas comunicar-se com outros via um metaobjeto que implementa um mecanismo de chamada de função específico. Mudanças no metaobjeto afetam a m

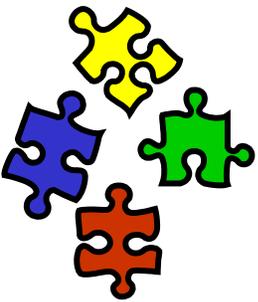




Estilo: Reflexão

- Uma interface é especificada para manipulação de metaobjetos. Esta interface é chamada **PROTOCOLO DO METAOBJETO** e é responsável por verificar a corretude da especificação de mudança e realizar a mudança

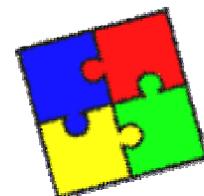


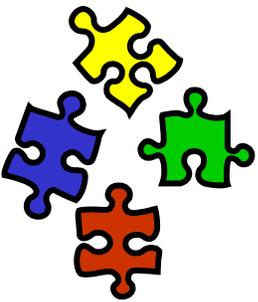


Estilo: Reflexão

Class Nível Base	Colaboradores Meta Nível	Class Meta Nível	Colaboradores Nível Base
Responsabilidade Implementar a lógica da aplicação Usar informação oferecida pelo meta nível		Responsabilidade Encapsula aspectos internos que podem mudar Oferece uma interface para facilitar modificações no meta-nível	

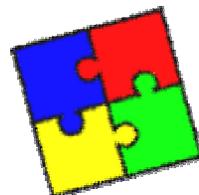
Class Protocolo do Metaobjeto	Colaboradores Meta Nível Nível Base
Responsabilidade Oferece uma interface para especificar mudanças no metanível Realiza as mudanças especificadas	

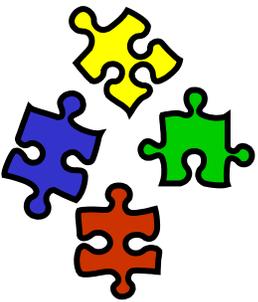




Estilo: Reflexão Implementação

- Defina um modelo da Aplicação
- Identifique o comportamento variante
 - Determine quais os serviços da aplicação que podem variar e quais permanecem estáveis – estes aspectos dependem do domínio da aplicação, dos seus usuários e do ambiente.
 - Exemplos de aspectos que podem variar:
 - Restrições Tempo-Real como deadlines, protocolos e algoritmos para detecção de dealines
 - Mecanismos de comunicação entre processos como RPC e Memória Compartilhada
 - Comportamento nos casos de exceções

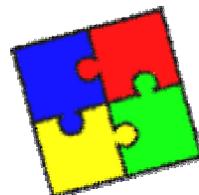


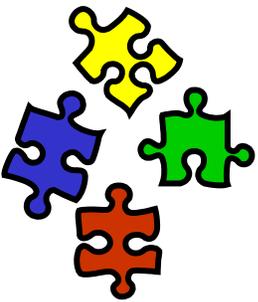


Estilo: Reflexão

Implementação - cont.

- Identifique os aspectos estruturais do sistema
 - Modelo de objetos, distribuição dos componentes, estrutura de tipos
- Identifique os serviços do sistema
- Defina os metaobjetos

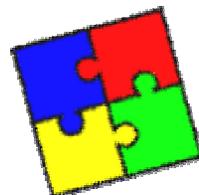


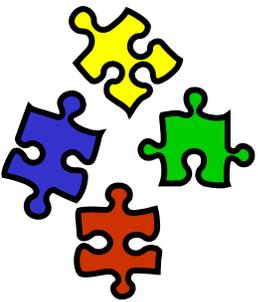


Estilo: Reflexão

Implementação - cont.

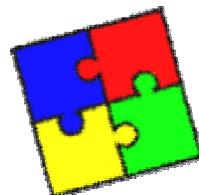
- Defina o protocolo dos metaobjetos
 - Formas de se implementar o protocolo
 - Integre-o com os metaobjetos. Cada metaobjeto oferece as funções do protocolo que opera sobre ele
 - Implemente o protocolo como um componente separado
- Defina o nível base
 - Use metaobjetos para tornar o nível base extensível e adaptável
 - Conecte cada componente do nível base com metaobjetos

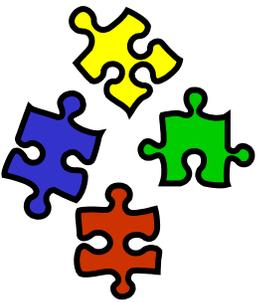




Estilo: Reflexão

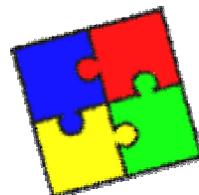
- Variações do Estilo:
 - *Reflexão com vários meta níveis*: algumas vezes metaobjetos dependem de outros. Para coordenar as dependências podem-se introduzir um meta meta nível

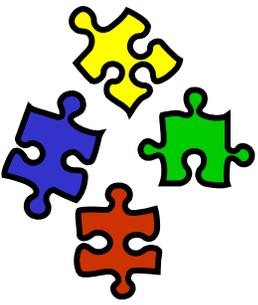




Estilo: Reflexão

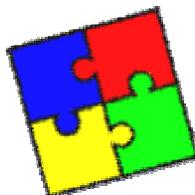
- CLOS (Common Lisp Object System) é uma linguagem de programação reflexiva onde operações que são definidas sobre objetos são chamadas **FUNÇÕES GENÉRICAS** e seu processamento é chamado **INVOCAÇÃO DA FUNÇÃO GENÉRICA**.
- Outras Ling. de Programação com suporte a reflexão: RbCl, Apertos e CodA

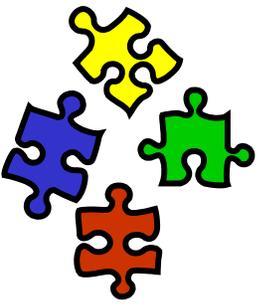




Estilo: Reflexão

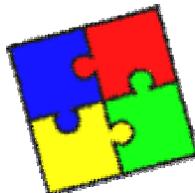
- Benefícios:
 - **Não há necessidade de modificação do código fonte:**
 - Mudanças são especificadas chamando uma função do protocolo do metaobjeto.
 - Para extensão do software o novo código é passado para o meta nível como um parâmetro do protocolo do metaobjeto. Tal protocolo é responsável por realizar as modificações e, se necessário recompilar as partes modificadas

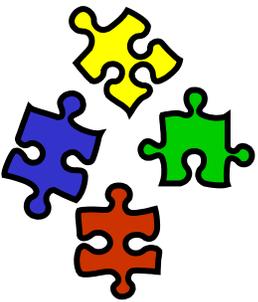




Estilo: Reflexão

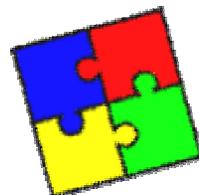
- Benefícios:
 - **Suporte a vários tipos de mudanças**
 - Metaobjetos podem encapsular diversos aspectos do comportamento do sistema, estado e estrutura. Portanto uma arquitetura baseada no estilo Reflexão pode suportar mudanças de vários tipos

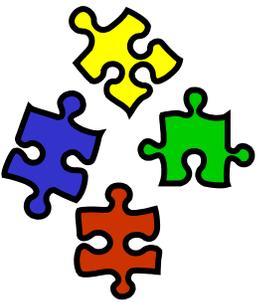




Estilo: Reflexão

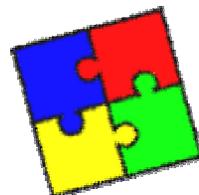
- Desvantagens:
 - **Nem todas as linguagens dão suporte a Reflexão:**
 - Em linguagens que não suportam reflexão como C++ é impossível explorar todo o poder da reflexão
 - **Baixa Eficiência:**
 - Sistemas reflexivos são mais lentos que sistemas não reflexivos devido a relação complexa entre o nível base e o meta nível

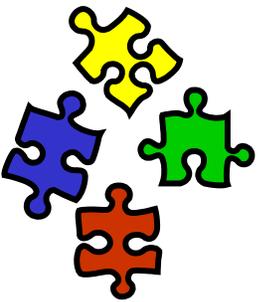




Exemplo de Aplicação do Estilo Reflexão

- Uma aplicação C++ que necessite de Persistência (escrever objetos em disco e ler objetos do disco)
 - Solução Clássica: implementar métodos com tipos específicos para armazenamento e leitura
 - Problema da solução clássica: sempre que a estrutura de classes da aplicação for alterada, os métodos devem ser modificados





Exemplo de Aplicação do Estilo Reflexão – cont.

- Solução com Reflexão:
 - Desenvolver um componente **PERSISTÊNCIA** – no nível base - que é independente de estruturas de dados específicas. No entanto, para armazenar e ler objetos C++ arbitrários é necessário acesso dinâmico a sua estrutura interna
 - Especificar metaobjetos que oferecem informações de tipo em tempo de execução
 - Especificar o protocolo de metaobjetos com uma função *getInfo* que permite os clientes acessarem informações sobre objetos

