Tratamento de erros: uma solução para usuários e desenvolvedores

Fernando Henrique Canto (UFRGS)
fernando@cpd.ufrgs.br

Clodoaldo de Borba Lambiase (UFRGS)
clodoaldo@cpd.ufrgs.br

Fernando Vieira Lázaro (UFRGS)
flazaro@cpd.ufrgs.br

Augusto Dias Pereira dos Santos (UFRGS)
asantos@cpd.ufrgs.br

Introdução

Este trabalho apresenta uma solução desenvolvida no CPD da UFRGS para tratamento e registro de erros e exceções em sistemas PHP. Serão explicados a motivação para o desenvolvimento dessa solução, os objetivos almejados, os recursos utilizados e os resultados obtidos.

Motivação

O desenvolvimento de sistemas de software é um processo extremamente suscetível a erros. Não importa o quão longas sejam as fases de análise, implementação e testes, o número de profissionais envolvidos e a sua competência e qualificação, não é possível garantir que um programa esteja 100% livre de bugs e incorreções. Dentre os motivos, pode-se listar o simples fato de que os desenvolvedores são humanos, e as ferramentas utilizadas não são perfeitas e possuem limitações. Além disso, não podemos prever perfeitamente a interação do usuário com o sistema; por mais intuitivo e amigável que o sistema seja, não se pode excluir a possibilidade de o usuário entrar dados inválidos (seja por erro ou para "testar" o sistema), realizar acessos indevidos a páginas Web (por exemplo, manipulando argumentos passados via endereços URL), ou apenas seguir uma linha de raciocínio que não foi prevista pelos analistas e desenvolvedores do sistema.

Por isso, por mais desagradável que possa parecer, um sistema deve ser desenvolvido com a convicção de que erros *certamente ocorrerão*. Existem boas práticas de programação e manuais que auxiliam o programador a fazer um programa bastante seguro, que não causa medo ou espanto ao usuário, nem realiza operações indevidas ou gera inconsistências na base de dados. Porém, sentimos a necessidade de termos um ambiente de trabalho onde os próprios recursos do PHP garantem um mínimo de segurança.

Além desse aspecto, há também a questão da depuração e correção desses bugs. Nem sempre é fácil decodificar as mensagens de erro recebidas, reproduzir o erro, localizar o arquivo

e a linha onde houve o erro, etc. Muitas vezes o processo de correção em si é extremamente simples (por exemplo, o erro pode consistir em um erro de digitação no nome de uma variável), mas a *localização* do bug pode ser demorada e frustrante. Isso resultou em um anseio ainda maior por um mecanismo que realizasse o tratamento desses erros, que fosse conveniente tanto para o usuário quanto para os desenvolvedores.

Na realidade, a necessidade desse mecanismo surgiu também devido à existência de um mecanismo semelhante, feito pelos desenvolvedores do CPD, para a linguagem ASP. Esse mecanismo realizava a redireção para uma página padrão, no caso de algum erro em tempo de execução, e permitia ao usuário enviar um e-mail para o analista responsável. Este sistema era extremamente útil, e com o tempo, novos recursos foram integrados a ele para facilitar o trabalho de depuração; por exemplo, no e-mail disparado para o analista, anexava-se os argumentos passados para a página, bem como as variáveis de sessão. Desejava-se criar um mecanismo semelhante para o PHP, e o primeiro passo seria procurar os recursos oferecidos pela linguagem que possibilitassem isso.

Erros vs. Exceções

Historicamente, a linguagem PHP sempre sinalizou erros de interpretação ou de execução através de um mecanismo próprio de erros. Divide-se esses erros em categorias (*notice*, *warning*, *parse error*, *fatal error*, etc.), cada uma com um significado específico. O desenvolvedor tinha a opção de habilitar ou desabilitar a exibição das mensagens de cada tipo de erro. Devido à natureza das mensagens *notice*, é comum desabilitá-las, porque certos vícios de programação (por exemplo, acessar variáveis não inicializadas, posições inexistentes de vetores, etc.) acabaram se tornando comuns. *Warnings* são problemáticos, pois geralmente indicam um problema na execução de alguma função (por exemplo, consultas SQL inválidas), mas não interrompem a execução do programa. O PHP também possui categorias para sinalizar erros fatais de execução, mas que não podem ser detectados no momento do *parsing* (por exemplo, divisões por 0, chamadas de métodos em objetos inválidos, etc.). Esses são chamados de *fatal errors*.

Exceções são um conceito herdado de linguagens orientadas a objetos. Java, por exemplo, funciona inteiramente usando exceções para sinalizar erros em tempo de execução. A ideia é que a classe *Exception* pode ser extendida em diversos tipos de exceção, cada um com seu significado. A linguagem, no caso de erro, *lança* uma exceção correspondente. Essa ação interrompe a execução do método, e vai se propagando pela pilha de execução até a exceção ser *capturada*, ou então até atingir a base da pilha e causar um erro fatal. Essa captura de exceções é feita utilizando estruturas de controle do tipo *try...catch*. Todo o código dentro da cláusula *try* fica "protegido" de exceções, e quando alguma ocorrer ali dentro, o tipo de exceção é procurado nas várias cláusulas *catch*, e ao ser encontrado, o trecho de código respectivo é executado, e o programa segue normalmente. Isso permite que o desenvolvedor faça um tratamento amigável de comportamentos incorretos ou inesperados, e também permite que ele faça classes e métodos independentes que não terão um comportamento inadequado se forem utilizados de maneira incorreta, protegendo porções críticas de código com exceções, que não podem ser burladas.

No PHP, as exceções foram incorporadas a partir da versão 5. Elas possuem duas grandes vantagens sobre os erros comuns: a possibilidade de uso das estruturas *try...catch*, sendo que não há equivalente para os erros; e o fato de que elas podem ser especializadas de forma mais

sofisticada, de acordo com a necessidade, através dos conceitos usuais da orientação a objetos, possibilitando assim um tratamento específico para cada tipo de erro.

Dentre os vários recursos de tratamento de erros que a linguagem PHP oferece, há duas funções muito poderosas que permitem ao programador desenvolver funções que *sobrecarregam* o tratador padrão para erros (*set_error_handler()*) e exceções (*set_exception_handler()*), e é justamente esse recurso que tornou possível o desenvolvimento deste trabalho.

A alternativa escolhida pelo nosso mecanismo de tratamento de erros consiste em fazer uma "conversão" de erros para exceções, utilizando a função *set_error_handler()* para lançar um tipo específico de exceção referente a um erro de linguagem. Isso foi decidido pois podemos criar diversos tipos de exceções, que podem ser tratadas de maneira específica, e também permitimos que os outros desenvolvedores criem exceções específicas para seus sistemas. Assim, não se trabalha com *erros* em um sistema PHP, possibilitando o uso de estruturas *try...catch* de maneira inteligente.

O mecanismo de tratamento

A implementação desse mecanismo é muito simples, e consistiu apenas em programar as funções a serem definidas como os tratadores (*handlers*) de erros e exceções, através das funções mencionadas acima. Isso resultou em um arquivo, chamado *Tratamento_Erros.php*, que deve ser incluído (através de *include* ou *require*) nos scripts onde se deseja utilizar esse tratamento. Embora essa obrigação pareça inconveniente, é comum o uso de outras bibliotecas padrão em diversos sistemas, portanto a inclusão do arquivo *Tratamento_Erros.php* pode ser feita nessas bibliotecas; além disso, não é necessário *nenhuma* alteração nos scripts originais para que o tratamento de erros funcione; ele não é intrusivo, e só é invocado nos casos de erro.

Como já foi dito, o *handler* de erros consiste apenas na detecção de alguns tipos específicos de erro (erro de execução, erro de consulta SQL, erro de envio de e-mail), e no lançamento de uma exceção a partir disso. O *handler* de exceções é onde ocorre o tratamento propriamente dito.

O tratamento realiza operações diferentes no ambiente de *testes* e no ambiente de *produção*. No primeiro ambiente, os dados relevantes daquele tipo de erro (mensagem, arquivo, linha, etc.) são exibidos na tela, juntamente com o *trace* de execução. No momento de testes, isso permite uma compreensão melhor e mais rápida do erro, porque até nesses casos pode não ser trivial descobrir a origem e o motivo do erro. Já no ambiente de produção, o *handler* exibe uma mensagem padronizada para o usuário final (fig. 1), seguindo os moldes e o *layout* das outras páginas, e com um botão de *Voltar*. Após isso, o *handler* realiza duas importantes operações para facilitar e agilizar a depuração de erros: a inserção de um registro em um *log* de erros, e o disparo de um e-mail para os responsáveis por aquele sistema.

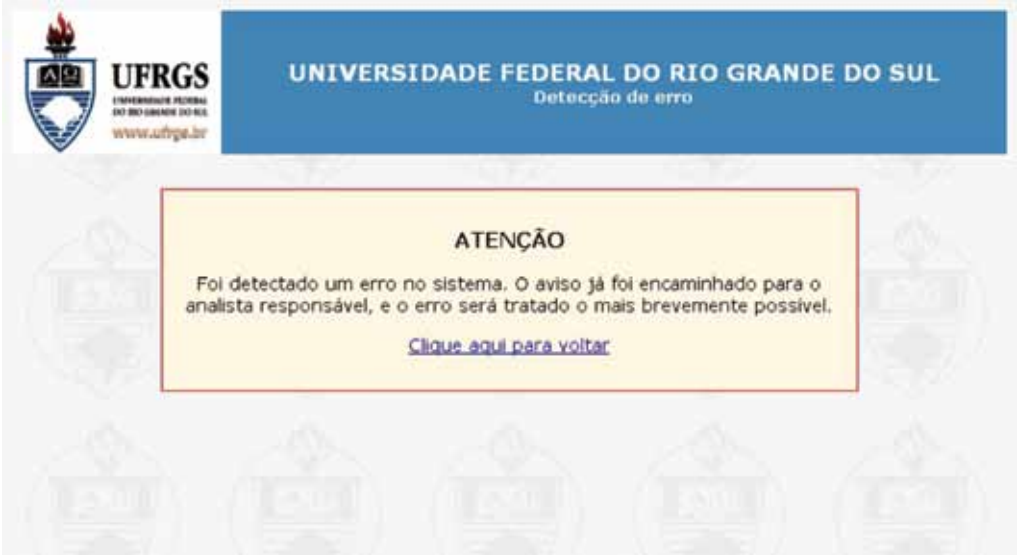


Figura 1: Mensagem exibida para o usuário

O *log* de erros

O Log consiste em uma tabela no banco de dados (tab. 1), que permite o registro dos erros ocorridos, em ordem cronológica, guardando tanto os dados mais relevantes quanto o detalhamento do *trace* e o contexto de execução (argumentos *POST* e *GET* e variáveis de sessão).

Após a exibição da mensagem padrão, o *handler* realiza uma operação *INSERT* nessa tabela. O caminho do arquivo onde ocorreu o erro, a linha, a mensagem, a data e a hora de ocorrência, o *backtrace* e os vetores de contexto (*$ POST*, *$ GET* e *$ SESSION*) são alguns dos registros inseridos. Através do arquivo, o tratador consulta uma tabela de responsáveis pelo sistema, recebendo de volta os endereços de e-mail do analista e do desenvolvedor responsável. Os dados relevantes são então enviados via e-mail para essas pessoas.

O Log é consultado em uma página restrita para os desenvolvedores e analistas. Através dessa tabela de responsabilidade, o usuário enxerga apenas os erros dos sistemas pelos quais é responsável. Esses erros são mostrados em ordem cronológica, e agrupados de acordo com a mensagem, o arquivo e a linha; isso faz com que um mesmo bug disparado várias vezes não apareça várias vezes na lista de erros, o que causaria poluição da lista.

A partir dessa lista, é possível acessar os detalhes de cada erro, onde são mostrados o *trace* e os vetores de contexto. O usuário, nesta tela, tem a possibilidade de marcar o erro como *solucionado* ou *não solucionado*, e fazer anotações pessoais que possam ser úteis em caso de ocorrência de erro semelhante em situações futuras.

Tabela 1: Esquema da tabela LOGERROSINTERNET

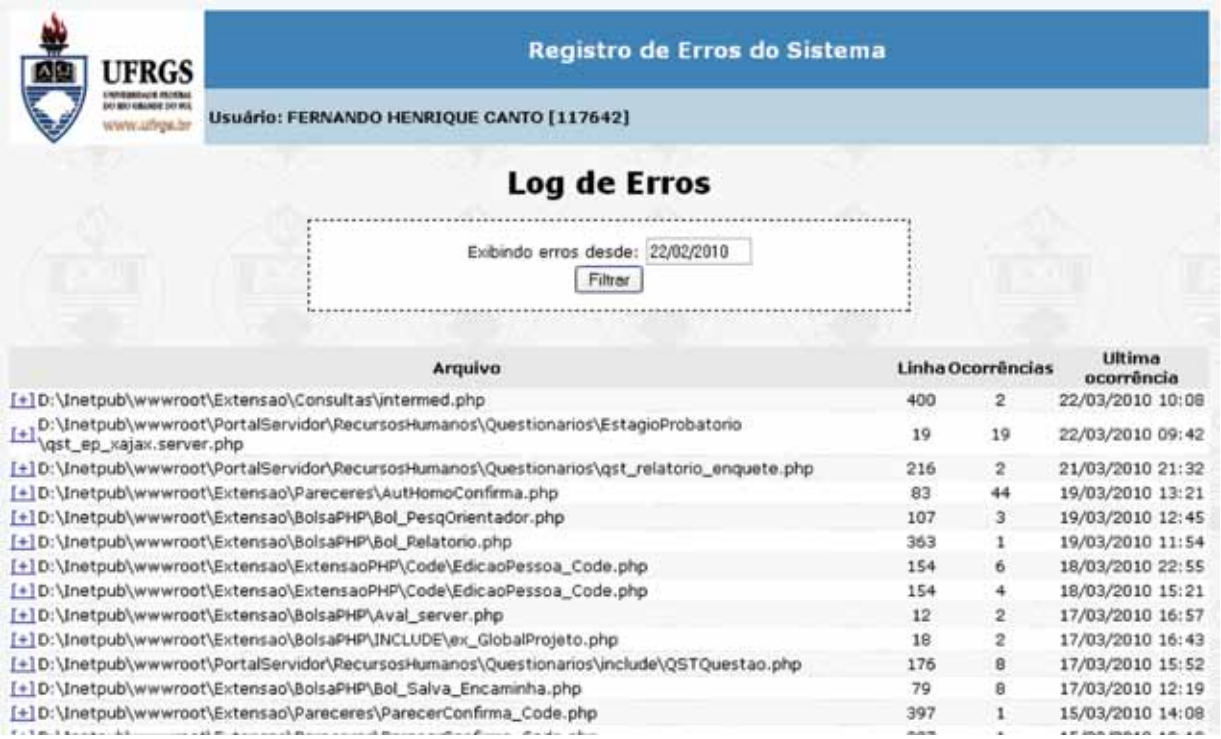
Nome da coluna	Tipo	Descrição
NrSeqLogErros	Número (PK)	Identificador único
NomeArquivoErro	Texto	Nome do arquivo onde ocorreu o erro
LinhaErro	Número	Número da linha do arquivo
MensagemErro	Texto	Mensagem de erro retornado pelo programa.
CodErro	Número	Código do erro; seu significado varia de acordo com o tipo de erro (erro de PHP, erro de SQL, exceção, etc.)
SQLErro	Texto	Consulta SQL que resultou no erro, se esse for o caso.
TracoErro	Texto	Traço de execução gerado pela linguagem, gravado como um vetor serializado.
DataGeracaoErro	Data	Data e hora da ocorrência do erro.
DataResolucaoErro	Data	Data e hora de resolução, preenchido pelo desenvolvedor
DescricaoProcedimentoResolucao	Texto	Texto explicativo opcional, preenchido pelo desenvolvedor.
VariaveisSessao	Texto	String serializada contendo os vetores de contexto de execução ($_SESSION, $_GET, $_POST)
PastaWEB	Texto	Diretório onde se encontra o arquivo. Utilizado para verificar os desenvolvedores e analistas que devem poder enxergar o erro.

Evolução da lista de erros registrados

Inicialmente, a lista de erros era extremamente simples, exibindo todos os erros em ordem cronológica descendente. A partir da lista, era possível ver o arquivo, a linha, data e hora de ocorrência do erro, um pedaço da mensagem de erro, e um link para a tela de detalhamento.

Com o uso mais frequente da lista, percebeu-se que seria necessário fazer mudanças na tela para permitir uma visualização mais confortável e eficiente. A primeira ideia era de agrupar os erros idênticos – isto é, os erros com a mesma mensagem, na mesma linha do mesmo arquivo – ordenando esses grupos em ordem cronológica descendente pela ocorrência mais recente. A segunda foi aplicar um filtro para exibir apenas os erros ocorridos dentro de um determinado período. A partir de um campo texto, é possível alterar esse período, para permitir a visualização de erros mais antigos. Isso aprimorou muito a interface e a interação; dentre outras coisas, se tornou muito mais simples detectar erros frequentes e provavelmente urgentes, e distingui-los de erros menos comuns. Além disso, o "encolhimento" da lista tornou-a mais funcional e mais eficiente (fig. 2).

A tela de detalhamento não precisou de alterações tão radicais; apenas, ao longo do tempo, viu-se a necessidade de incluir os vetores de contexto, que antes não eram exibidos nem guardados na tabela. Percebeu-se que, muitas vezes, o erro era causado pela ausência ou inconsistência de um argumento recebido pela página, o que tornava necessário rastrear precisamente a origem daquela inconsistência. Sabendo exatamente qual é a inconsistência, a tarefa de encontrá-la e corrigi-la tornou-se bem mais rápida.



Arquivo	Linha	Ocorrências	Ultima ocorrência
[+] D:\Inetpub\wwwroot\Extensao\Consultas\intermed.php	400	2	22/03/2010 10:08
[+] D:\Inetpub\wwwroot\PortalServidor\RecursosHumanos\Questionarios\EstagioProbatorio\qst_ep_xajax.server.php	19	19	22/03/2010 09:42
[+] D:\Inetpub\wwwroot\PortalServidor\RecursosHumanos\Questionarios\qst_relatorio_enquete.php	216	2	21/03/2010 21:32
[+] D:\Inetpub\wwwroot\Extensao\Pareceres\AutHomoConfirma.php	83	44	19/03/2010 13:21
[+] D:\Inetpub\wwwroot\Extensao\BolsaPHP\Bol_PesqOrientador.php	107	3	19/03/2010 12:45
[+] D:\Inetpub\wwwroot\Extensao\BolsaPHP\Bol_Relatorio.php	363	1	19/03/2010 11:54
[+] D:\Inetpub\wwwroot\Extensao\ExtensaoPHP\Code\EdicaoPessoa_Code.php	154	6	18/03/2010 22:55
[+] D:\Inetpub\wwwroot\Extensao\ExtensaoPHP\Code\EdicaoPessoa_Code.php	154	4	18/03/2010 15:21
[+] D:\Inetpub\wwwroot\Extensao\BolsaPHP\Aval_server.php	12	2	17/03/2010 16:57
[+] D:\Inetpub\wwwroot\Extensao\BolsaPHP\INCLUDE\ex_GlobalProjeto.php	18	2	17/03/2010 16:43
[+] D:\Inetpub\wwwroot\PortalServidor\RecursosHumanos\Questionarios\include\QSTQuestao.php	176	8	17/03/2010 15:52
[+] D:\Inetpub\wwwroot\Extensao\BolsaPHP\Bol_Salva_Encaminha.php	79	8	17/03/2010 12:19
[+] D:\Inetpub\wwwroot\Extensao\Pareceres\ParecerConfirma_Code.php	397	1	15/03/2010 14:08

Figura 2: Log de erros

O envio de e-mails sempre foi um recurso essencial, pois com ele pode-se dar uma resposta muito mais rápida ao problema. Tanto o e-mail quando o log são importantes e complementares, e o mecanismo se tornaria bastante incompleto com a ausência de um deles.

Tipos de erros

O uso de exceções no PHP permitiu a categorização dos erros, fazendo, na prática, com que o mecanismo de tratamento fizesse uma operação diferente com cada um deles. Listamos e explicamos, abaixo, os tipos mais relevantes e sua serventia:

- *ErrorException*: trata-se de um erro gerado pela linguagem PHP, que é "traduzido" em uma exceção. Essa tradução é feita apenas para simplificar o mecanismo e evitar redundâncias no código, pois a exceção não contém nenhum recurso que seja impossível de se recriar com erros comuns (o *trace* gerado pela exceção, por exemplo, pode ser obtido a qualquer momento com uma chamada a uma função padrão da biblioteca PHP). Geralmente, só se capturam erros fatais de execução, o que resulta na interrupção imediata da execução do script, a exibição da mensagem padrão de erro, e o registro na tabela de Log. É perfeitamente possível ativar o mesmo comportamento para erros do tipo *Warning* e *Notice*.
- *SQLException*: é usada especificamente para tratar de erros em consultas SQL. Estes erros recebem o mesmo tratamento dos erros acima. É necessário tratar erros de SQL de maneira específica, pois as mensagens geradas pela linguagem são apenas *warnings*. Fora isso, o fato de usarmos uma biblioteca própria para conexão com o banco de dados (DBPHP) faz com que os erros sejam gerados pelas chamadas de função dentro da

própria biblioteca, quando na verdade o importante é saber qual foi o arquivo e a linha que invocou a biblioteca.

- *Exception*: exceções comuns são tratados do mesmo jeito. Com isso, é possível para o programador identificar e isolar comportamentos inadequados dentro de seu código, fazendo com que o próprio tratamento de erros tome as devidas providências. Isso é útil quando, por exemplo, um desenvolvedor cria uma classe, e quer garantir que os argumentos passados para os métodos sejam todos consistentes. Ao invés de permitir que o método tenha um comportamento incorreto, o script pode testar a integridade dos dados recebidos, lançando uma exceção em caso de erro.
- *EMailException*: esta exceção é lançada quando o erro é disparado pela função *mail()* da biblioteca PHP. Isso é útil para permitir a identificação de endereços inválidos na base de dados, por exemplo. Neste caso, a execução do script *não* é interrompida, mas o desenvolvedor pode usar uma estrutura *try...catch* para tratar casos como esse.
- *UserException*: esta exceção foi feita para uso dos desenvolvedores. Ela permite que o programador defina a mensagem que será exibida para o usuário quando ela for lançada, servindo assim de aviso sobre uma ação incorreta ou inesperada. Neste caso, a execução será interrompida.

É importante lembrar que a execução do script original somente será interrompida caso a exceção não for tratada dentro de um *try...catch*.

O uso da instrução *throw*

O mecanismo de tratamento de erros é feito para tratar exceções de maneira geral, com comportamentos especiais apenas para os tipos de exceção listados acima. Portanto, qualquer exceção lançada por um script será tratada da maneira genérica. Isso em geral é bom, mas seu uso requer alguns cuidados.

As boas práticas de programação dizem que as exceções devem ser reservadas para casos realmente excepcionais, de uso realmente incorreto dos recursos oferecidos. Não é recomendado aplicar exceções em casos corriqueiros; por exemplo, em consistência de dados fornecidos via um formulário. O ideal é, nesses casos, utilizar recursos mais usuais. O uso inadequado de exceções pode resultar em mensagens de erro indesejadas, quando não forem devidamente tratadas com cláusulas *try...catch*.

Porém, quando bem utilizadas, as exceções podem ser um excelente recurso. Com o seu uso, o programador pode proteger o código de comportamentos indesejados, sem se preocupar com a exibição da mensagem de erro, e também tendo a possibilidade de rastrear a ocorrência desses casos inválidos.

Um caso notável que presenciamos foi em um sistema de inscrição, que envolvia submissão de trabalhos no formato PDF. Para fins de visualização, uma página PHP recebia o código do trabalho como um argumento passado via GET e devolvia o arquivo PHP correspondente. Porém, por fins de segurança, o sistema verificava se o trabalho realmente pertencia à pessoa que estava logada, e na tentativa de acesso a um trabalho inválido, a página lançava uma exceção. Olhando os logs de erro, percebemos uma série de tentativas de acesso inválidas, realizadas por um usuário, feitas alterando o valor do argumento na linha de endereço. O uso de exceções

não só resolveu uma questão de segurança de dados, mas mostrou um exemplo prático da necessidade de proteger dados sensíveis, ao invés de confiar na ingenuidade do usuário.

Planos futuros

Este recurso ainda tem algumas expansões e alterações planejadas, porém não executadas. Uma delas é referente às permissões e responsabilidades sobre sistemas; no momento, o sistema apenas prevê um analista e um desenvolvedor por diretório, quando seria mais correto atribuir responsabilidades a N usuários através de uma tabela específica para isso.

Um problema ainda não resolvido ocorre quando o tratamento de erros é usado em um script PHP que é requisitado via AJAX. Nesse caso, não faz sentido mostrar a tela padrão, pois a saída do script é recebida por uma função JavaScript que deveria estar preparada para tratá-la. Este problema se torna bastante complexo em sistemas que usam o AJAX "puro", pois o programador implementa tanto o script PHP quanto a função JavaScript que recebe a resposta, e a alteração dessa resposta no caso de erros pode ter efeitos imprevisíveis sobre sistemas já prontos. Quando se usa uma biblioteca AJAX – como o *xajax*, por exemplo – a solução se torna um pouco mais simples, pois raramente é necessário implementar o código JavaScript; mas esta solução ainda está sob análise.

Conclusão

Com o seu uso cada vez mais frequente, o Tratamento de Erros se tornou um dos maiores ajudantes na tarefa de depurar e corrigir falhas no sistema. A facilidade de integração do mecanismo em páginas já existentes, combinada com sua praticidade, conveniência e simplicidade, faz do uso dessa ferramenta uma prática quase que sem desvantagens. É realmente revelador constatar que uma solução como essa possa tornar mais fácil e segura a vida dos usuários, dos analistas e dos programadores, e mostra que não é necessário sacrificar um desses lados para privilegiar os outros.

Outro detalhe importante deste mecanismo é que ele é muito simples de incorporar em outros ambientes, até mesmo ambientes bastante grandes e complexos, com mínimas adaptações. A simplicidade e a facilidade de uso dessa ferramenta tornam ainda mais satisfatórias as vantagens que ela proporciona.