Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Optimizações guiadas por dados reais em Linguagens de Domínio Específico para a Web

Miguel Alexandre Figueira Rebelo — 31224

Lisboa
(2011)

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Optimizações guiadas por dados reais em Linguagens de Domínio Específico para a Web

Miguel Alexandre Figueira Rebelo — 31224

Orientador: Prof. Doutor Luís Caires
Co-orientador: Eng. Lúcio Ferrão

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para a obtenção do Grau de Mestre em Engenharia Informática.

Lisboa
(2011)

Para o meu pai Carlos,
a minha mãe Fátima,
os meus irmãos Diogo e Tiago
e o meu amor Elza.
Obrigado por tudo.

Agradecimentos

Começo por agradecer aos meus coordenadores, Luís Caires e Lúcio Ferrão, pelo apoio prestado ao longo de toda esta jornada. A vossa ajuda foi essencial para atingir os objectivos deste projecto. Agradeço também ao Professor João Costa Seco por me ter aberto as portas para esta oportunidade, tal como o apoio prestado sempre que foi necessário.

Agradeço a todos os meus colegas da equipa OutSystems que tornaram possível a realização deste projecto. Hélio Dolores e Hugo Veiga, agradeço-vos todo o apoio dado durante a fase inicial desta jornada, tal como a disponibilidade mostrada para ajudar sempre que fosse necessário ao longo de todo o projecto. Vocês foram essenciais, principalmente na fase de integração na equipa. Lúcio Ferrão e Miguel Ventura, por todo o tempo que vos roubei para se dedicarem ao meu projecto, deixo-vos aqui o meu grande Obrigado!

Tiago Leão, Eduardo Marques, Valter Balegas, Gonçalo Lucas, Roberto Silva, Frederic Veiga, João Loureiro, Tiago Santos, Pedro Borges, Vasco Pessanha, João Saramago, Ricardo Alves, João Vaz, André Guerreiro e Luís Miguel, obrigado por todos os momentos que passámos durante os vários anos de faculdade. Transformaram este percurso numa agradável viagem.

A todos os meus amigos que me acompanham desde sempre, André Silva, Daniel Grou, Jonathan Cuendet, Gonçalo Lucas, Eduardo Correia, André Serafim e Mauro Toscano. Obrigado por todos os bons momentos de descontracção e alegria que passámos e que vamos continuar a passar.

Finalmente, quero agradecer à Elza Rodrigues por todos os momentos que passámos juntos. Desde à muito tempo que és a minha amiga, a minha conselheira, a minha audiência na preparação de apresentações, mas acima de tudo, o meu grande amor. Ninguém me conhece melhor do que tu. Aos meus pais, Carlos e Fátima, e aos meus irmãos, Diogo e Tiago, obrigado por todo o amor e carinho que sempre me deram.

A todos vocês deixo aqui o meu Muito Obrigado por tudo.

Sumário

A utilização de linguagens para domínios específicos tendo em vista um processo de desenvolvimento mais simples e eficiente é um padrão cada vez mais frequente no desenvolvimento de aplicações para a web. Apesar destas linguagens permitirem aos programadores direccionarem o foco de desenvolvimento para as funcionalidades das suas aplicações, estas tendem em produzir aplicações pouco cuidadas a nível do consumo de recursos críticos, limitando a escalabilidade das mesmas. A forma típica de endereçar estas limitações passa por, após o desenvolvimento das suas aplicações, os programadores submeterem as mesmas a testes de desempenho e de carga recorrendo a ferramentas que lhes permitem simular ambientes de utilização reais. Contudo, simular um ambiente idêntico ao que será posteriormente encontrado em produção é um desafio que requer um investimento muito significativo em tempo, competências e atenção por parte da equipa de desenvolvimento, competindo por isso por recursos já escassos no seio de uma organização não dedicada ao desenvolvimento web. Isto leva a que normalmente os programadores apenas consigam identificar os problemas das suas aplicações quando estas, já em execução, começam a falhar. Um dos pontos críticos mais frequentes encontrados em aplicações web é a centralização dos dados em bases de dados, sendo criada uma grande carga sobre o servidor que mantém esses dados.

Esta dissertação apresenta uma solução de baixo custo e com ganhos significativos que permite, através da análise de dados reais recolhidos durante a execução das aplicações, guiar os compiladores e os programadores a identificarem as entidades dos modelos de dados das aplicações cujos acessos podem ou devem ser optimizados através de mecanismos de cache.

Com esta solução disponibilizámos aos programadores um método fácil e eficiente de identificarem as entidades candidatas à utilização de cache e introduzirem tais optimizações através de mecanismos simples. O risco associado à introdução de cache com a nossa solução é muito baixo, pois foi criado um mecanismo de invalidação que permite que os dados mantidos em cache sejam invalidados sempre que os dados que estes dependem sofram alterações na base de dados.

A nossa solução foi implementada e testada sobre um exemplo concreto de uma DSL de alto nível, desenvolvida pela empresa *OutSystems*, donde se obtiveram resultados que indicam ganhos significativos no desempenho das aplicações finais, melhorando assim a experiência dos utilizadores finais dessas aplicações. Um dos principais objectivos desta solução era dimi-

nuir a carga existente sobre o servidor de bases de dados. Este objectivo foi sucedido, tendo sido obtida uma diminuição de pedidos sobre o servidor de base de dados de cerca de 60% (teste realizado em ambiente simulado).

Palavras-chave: Optimização baseada em Profiling, Optimização em compiladores.

Abstract

The use of domain-specific languages in order to make the development process easier and more efficient is an increasingly common pattern in developing web applications. These languages allow developers to target the focus of development in the application functionality, which usually leads to the development of applications without concerns about consumption of critical resources, restricting the application scalability. The typical way of addressing these limitations is by submitting applications to performance and load tests, using tools that allow to simulate production environments. However, to simulate an environment similar to what will later be found in production is a challenge that requires a significant investment in time, skill and attention by the development team, so competing for already scarce resources within an organization not dedicated to web development. This usually means that developers are only able to identify the problems of their applications when, already in production, they begin to fail. One of the most common critical points found in web applications is the centralization of data in databases, due to the large load on the server database.

This dissertation presents a low cost solution with significant gains that allows the compiler and the developer to identify the entities of the application data model whose accesses can and should be optimized through caching patterns. To identify such entities this process is guided by usage data captured in production environment.

With this solution, we provided to the developers an easy and efficient way to identify the candidate entities to use caching patterns, and to optimize their applications using simple mechanisms. To reduce the risk of inconsistency, we created an invalidation mechanism that allows data held in cache is invalidated when the data they depend are changed in the database.

Our solution was implemented and tested in top of a high level domain-specific language, developed by the company *OutSystems*, from which we obtained results that show significant gains in the application performance, thus improving the user experience. One of the main goals of this solution is to reduce unnecessary load on the server database. This goal was successful, having been obtained a decrease in amount of requests on the database server of about 60% (test performed in a simulated environment).

Keywords: Profiling based optimizations, Optimizations in compilers.

Conteúdo

Lista de Figuras

Lista de Tabelas

Listagens de Código

Contexto do Trabalho

A elaboração deste trabalho foi realizada em ambiente empresarial, na empresa *OutSystems*, tendo sido integrado na equipa de Investigação e Desenvolvimento.

Durante a primeira fase deste trabalho realizámos um estudo preliminar e desenhámos uma possível solução para o nosso sistema de optimização e sugestões de optimizações baseados em dados reais de utilização das aplicações. Nesta fase estudámos vários tipos de optimizações possíveis de implementar, tendo sido seleccionada a optimização de introdução de mecanismos de cache sobre entidades da base de dados por ser aquela onde previmos que se teriam maiores ganhos no desempenho das aplicações.

Para que na fase de desenvolvimento nos pudéssemos focar na implementação da nossa solução, foi necessário entender as várias componentes existentes na plataforma *OutSystems*, plataforma sobre a qual foi implementada a nossa solução. Nesta fase não só foi importante compreender as funcionalidades de cada componente da plataforma, como também o modo como estas estavam implementadas.

No decorrer da fase inicial do projecto foram discutidos com pormenor os vários componentes da solução proposta, o que permitiu que na fase de desenvolvimento nos focássemos apenas na implementação, componente a componente, da solução idealizada. Durante toda a elaboração deste trabalho foi essencial a discussão e troca de ideias com vários elementos da equipa *OutSystems*, o que permitiu que este trabalho tivesse como resultado final uma solução completamente funcional e pronta para ser introduzida numa nova versão da plataforma.

Apesar deste projecto ser individual foi indispensável o apoio dado pela equipa *OutSystems*, tanto na discussão de ideias que permitiram chegar a uma melhor solução, como na transferência de conhecimento acerca dos vários componentes existentes na plataforma *OutSystems*, permitindo assim uma fácil integração numa plataforma madura, com mais de um milhão de linhas de código.

1

Introdução

O objectivo deste trabalho é desenvolver uma solução que possibilite a optimização e sugestão de optimizações às aplicações web, através da análise de dados estatísticos recolhidos durante a execução dessas aplicações.

Na secção seguinte vamos apresentar o contexto em que este trabalho se insere e quais as principais motivações para se seguir este tipo de abordagem. A motivação surge no contexto concreto de um sistema real, a *Agile Platform*, produzida pela empresa multinacional baseada em Portugal, *OutSystems SA*, com a qual o grupo de investigação *Centro de Informática e Tecnologias da Informação*, do departamento de informática da *Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa* mantém colaborações em actividades de *Investigação e Desenvolvimento*.

1.1 Contexto e Motivação do Problema

As linguagens de domínio específico (DSL) são linguagens de programação adaptadas a um domínio restrito de problemas. Este tipo de linguagens, devido à sua focalização num determinado domínio, tendem a ser muito eficientes no desenvolvimento de aplicações à medida [15], colocando o foco do desenvolvimento em níveis de abstracção mais próximos do negócio e mais distantes dos detalhes de implementação. Contudo, este foco leva a que os programadores desenvolvam aplicações funcionalmente correctas, mas pouco cuidadas a nível do consumo de recursos críticos, limitando a escalabilidade das mesmas. Este aspecto apenas se torna relevante quando uma aplicação em produção apresenta uma degradação da experiência dos utilizadores finais, o que normalmente acontece muito após o desenvolvimento das funcionalidades das mesmas.

Para endereçar este problema, os programadores têm de identificar os pontos da aplicação

que acusam um fraco desempenho (pontos críticos) e melhorar esses mesmos pontos através da introdução de optimizações. Contudo, este processo de identificação e optimização de pontos críticos, implica que os programadores que deveriam ter como foco apenas a funcionalidade das aplicações, tenham de sair desse nível de abstracção para analisar detalhes de implementação. Para facilitar o processo de detecção de pontos críticos, alguns sistemas possuem mecanismos de *profiling* [1] [5, 19].

A principal motivação para este trabalho incide na grande oportunidade de optimizações que podem ser introduzidas neste tipo de linguagens para que os programadores possam realmente focar-se no desenvolvimento das funcionalidades das aplicações, deixando os problemas de optimização a cargo dos compiladores [2, 3, 28, 35] e dos ambientes de suporte à execução das aplicações.

1.2 Objectivos da Dissertação

Nesta secção são apresentados os principais objectivos que pretendemos atingir com a nossa solução, sendo de seguida apresentadas as dificuldades encontradas pelos programadores durante o processo de identificação e optimização dos pontos críticos da sua aplicação. Por último apresentamos um estudo realizado sobre uma aplicação web empresarial real, que nos permitiu identificar concretamente problemas de optimização que podemos encontrar neste tipo de aplicações.

Na secção anterior, apresentámos alguns problemas que surgem em sistemas onde o principal foco dos programadores está direccionado para as funcionalidades das aplicações e não para os seus detalhes de implementação. Esta aproximação por vezes traduz-se no desenvolvimento de aplicações não optimizadas, cujo desempenho é desnecessariamente baixo, reflectindo-se no aumento do tempo de espera dos utilizadores.

O objectivo deste trabalho é desenvolver uma solução que possibilite a optimização e sugestão de optimizações às aplicações web, guiada pela análise de dados estatísticos recolhidos durante a execução dessas aplicações. Com esta arquitectura espera-se aumentar a usabilidade das aplicações, reduzindo o tempo de espera dos utilizadores pelas respostas do servidor aos seus pedidos [29].

Queremos manter o foco dos programadores no desenho funcional das aplicações, ajudando-os a endereçar trivialmente os desafios de escalabilidade e usabilidade das mesmas. Para isso, pretendemos implementar uma solução que permita à plataforma de desenvolvimento de aplicações sugerir melhoramentos às mesmas, para que os programadores os possam aplicar com um custo e um risco, ao nível do comportamento das aplicações, muito baixo para os ganhos envolvidos. Para os casos em que os riscos são bastante baixos e os ganhos que se irão obter a nível do desempenho das aplicação são óbvios, o compilador pode aplicar directamente as sugestões sem que seja necessária a aprovação do programador.

Por último, pretende-se realizar um estudo para avaliar o impacto real deste sistema em aplicações desenvolvidas com a *Agile Platform* da *OutSystems* [30], um caso específico de uma

[1] Processo de recolha de dados estatísticos sobre a execução das aplicações durante a execução das mesmas.

plataforma que utiliza uma linguagem de domínio específico.

Para entendermos melhor a importância que a solução que se pretende conceber tem na optimização de aplicações, de seguida é apresentado o procedimento realizado pelos programadores no processo de identificação e optimização de pontos críticos das aplicações quando estes se encontram em ambientes de desenvolvimento tradicionais, onde não existe suporte do lado da plataforma para os guiar neste processo.

1.2.1 Identificação e Optimização de Pontos Críticos

Quando se desenvolvem aplicações web, os programadores têm à sua disposição ferramentas que lhes permitem realizar testes de desempenho em ambientes simulados. Contudo, simular ambientes idênticos a ambientes reais é só por si um desafio [24, 25], que requer um investimento muito significativo em tempo, competências e atenção por parte da equipa de desenvolvimento, competindo por isso por recursos já escassos. Esta dificuldade encontrada na fase de testes normalmente leva a que os programadores não tenham conhecimento real sobre o desempenho final dessas aplicações até ao momento em que estas são colocadas em produção, tornando-se acessíveis aos utilizadores finais. Este facto leva a que alguns pontos críticos das aplicações apenas sejam conhecidos no momento em que esta começa a obter um fraco desempenho na resposta aos pedidos dos utilizadores.

Para facilitar o processo de identificação dos pontos críticos das aplicações, o programador pode seguir duas abordagens diferentes. A primeira corresponde a recorrer a mecanismos de medição e instrumentação manual do código que lhe disponibilizem a informação necessária à identificação dos pontos críticos. A segunda abordagem consiste em esperar que os utilizadores lhe transmitam algum *feedback* que possa ser útil neste processo.

Tendo sido identificadas as operações com fraco desempenho o programador pode então realizar optimizações nesses mesmos pontos.

Tipicamente, o programador pode realizar optimizações em dois níveis, ao nível da interface do utilizador e ao nível dos acessos a dados. Uma optimização que é normalmente considerada, devido ao impacto que normalmente introduz no desempenho das aplicações, é a introdução de cache [13, 20, 32]. Esta optimização consiste em colocar uma cópia temporária dos dados usados com maior frequência num local mais próximo de quem lhes pretende aceder, possibilitando um acesso mais rápido a estes. O custo desta optimização prende-se ao mecanismo de invalidação que é necessário implementar para garantir a consistência entre as cópias dos dados e os seus originais.

Seleccionado o tipo de optimização a introduzir às aplicações, o programador introduz progressivamente cache nos pontos que este considerou como críticos, analisando o desempenho das aplicações após essas optimizações. Caso essas optimizações não se traduzam em melhorias suficientes no desempenho das aplicações, os programadores devem continuar à procura de novos pontos de optimização.

Este processo de identificação e optimização de pontos críticos é um processo que exige bastante trabalho por parte dos programadores mas que não pode ser ignorado, pois é completa-

mente essencial para melhorar o desempenho e consequentemente a usabilidade das aplicações desenvolvidas [29].

Como um dos objectivos deste trabalho é a sua aplicação a ambientes reais, realizámos um estudo sobre uma aplicação web empresarial real que será utilizada mais tarde para validação deste trabalho.

1.2.2 Exemplo Baseado na Análise de um Caso Real

Para que numa fase inicial do nosso trabalho pudéssemos entender melhor o impacto que os problemas atrás idenfiticados têm em ambientes reais, decidimos realizar um estudo sobre uma aplicação web empresarial que pudesse ser utilizada como aplicação padrão. Na escolha da aplicação a ser utilizada como caso de estudo tivemos em conta os seguintes factores que caracterizam grande parte das aplicações web empresariais: feita à medida; mantida por diferentes pessoas durante intervalos muito pequenos de tempo; ter alguma idade (+2 anos); ser de uso interno e ser centrada nos seus dados em base de dados. Analisando estes factores decidimos utilizar como caso de estudo o *IssuesManager*. Para suportar este estudo recorremos a um programador experiente envolvido no desenvolvimento da aplicação e com um conhecimento profundo do seu modelo de dados.

O *IssuesManager* é um exemplo duma aplicação web empresarial, que disponibiliza funcionalidades de gestão de projectos para desenvolvimento de software. Esta aplicação é utilizada por várias equipas, cada uma com um número máximo de 6 pessoas. Esta é uma aplicação crítica para as equipas que a utilizam, tendo sido desenvolvida ao longo do tempo, sem preocupações de custos de manutenção futuros nem preocupações de usabilidade por ser usada e desenvolvida apenas internamente à empresa.

Um conceito presente nesta aplicação é o conceito de tarefa, sendo a funcionalidade usada com mais frequência a de edição de uma tarefa. Esta funcionalidade é disponibilizada através de uma página que contém um formulário para ser preenchido pelo utilizador.

Ao analisarmos a página de edição de tarefas da aplicação *IssuesManager*, observamos que uma parte significativa da informação apresentada na página é obtida através de entidades [2] cujos valores se mantêm inalterados durante longos períodos de tempo. Após a análise do conjunto de entidades usado nessa página, suportada exclusivamente pela intuição e conhecimento profundo de um programador experiente envolvido no desenvolvimento da aplicação, previmos que do conjunto de 29 entidades utilizadas na criação da página, os dados guardados em 10 dessas entidades (cerca de 35% do conjunto total) mantêm-se inalterados por vários meses, como é o caso das entidades STATUS e KIND, que representam o estado e o tipo das tarefas, respectivamente. Analisando as consultas efectuadas à base de dados, observamos que dum conjunto total de 44 consultas, 14 são consultas realizadas exclusivamente sobre o conjunto de 10 entidades anteriormente identificado, o que representa cerca de 32% do total de consultas efectuadas.

Apesar de cerca de 32% das consultas à base de dados realizadas no carregamento da página

[2] O termo *entidade* é utilizado ao longo do documento para nos referirmos a entidades do modelo de dados.

produzirem praticamente sempre o mesmo resultado entre pedidos diferentes (com excepção quando se realiza uma actualização em alguma das entidades intervenientes nas consultas, o que acontece de modo pouco frequente), sempre que um utilizador realiza um pedido sobre esta página, é necessário efectuar novamente essas consultas. Cada uma destas consultas tem associado não só o custo temporal de se obterem os dados pretendidos no servidor de bases de dados, como também acresce o custo introduzido pela latência na rede durante a comunicação entre os servidores [31].

O padrão identificado no exemplo anterior tende em repetir-se por várias aplicações web empresariais desenvolvidas com linguagens de domínio específico, o que nos permite concluir que este é um possível ponto de optimização que se pode explorar nestas linguagens, de modo a melhorar o desempenho das aplicações reduzindo a quantidade de acessos a bases de dados, melhorando assim a usabilidade dessas aplicações [29].

Apesar do estudo realizado com o apoio de um programador experiente ter evidenciado um ponto de optimização com possíveis ganhos de desempenho, acreditamos que se tivessemos em nossa posse dados reais sobre a utilização de cada entidade, a opinião do mesmo programador poderia ser diferente, assim como o conjunto de entidades identificadas.

1.2.3 Factores que Influenciam a Decisão de Optimização

Analisando o estudo atrás apresentado há uma questão que rapidamente nos ocorre:

- Se é óbvio para o programador que existe um conjunto de acessos à base de dados que produz sempre os mesmos resultados porque é que o programador não optimiza esses pontos manualmente?

Existem vários factores que levam a que o programador não tome a iniciativa de introduzir optimizações nesses pontos das aplicações, como por exemplo:

Falta de pressão - a inexistência de algo ou alguém que indique ao programador que determinada optimização deve ser realizada leva a que este não sinta pressão para introduzir tal optimizar nas suas aplicações.

Falta de contexto - por vezes os programadores não têm acesso a informação útil sobre a utilização das suas aplicações, o que leva a que estes não sintam confiança sobre o verdadeiro impacto que as optimizações a introduzir possam ter no desempenho das aplicações.

Risco - algumas optimizações têm um risco associado o que pode retirar confiança ao programador no momento de decisão sobre introduzir ou não tais optimizações. Um exemplo de uma optimização com risco associado é a utilização de mecanismos de cache sobre os acessos à base de dados, onde está presente o risco de inconsistência entre os dados na base de dados e na cache.

Esforço - a optimização de aplicações pode tornar-se uma tarefa complicada, exigindo recursos que nem sempre se encontram disponíveis.

Posto isto torna-se mais claro que apesar do programador considerar que uma determinada optimização iria possivelmente melhorar o desempenho das suas aplicações, existem vários factores que levam a que este não avance. Com o nosso trabalho tentámos abordar todos estes factores de modo a que o programador possa optimizar as suas aplicações com confiança.

1.3 Desafios

O problema que se propõe tratar nesta dissertação, requer a combinação de várias técnicas avançadas de engenharia informática, assim como um esforço de concepção onde se espera poder introduzir alguma inovação.

Para se conceber a solução pretendida de modo a que esta satisfaça os objectivos atrás definidos, surgem vários desafios, dos quais destacamos aqueles com maior importância:

- Para determinar que optimizações devem ser sugeridas ou aplicadas directamente às aplicações é necessário obter-se informação sobre a execução dessas aplicações. Como podem esses dados ser recolhidos sem causar um impacto significativo no tempo de execução dessas aplicações?

- As sugestões dadas pela plataforma devem ser úteis para o programador. Como poderá uma solução como a que se pretende implementar, decidir sobre que alterações irão efectivamente melhorar o desempenho das aplicações?

- O uso das optimizações introduzidas deve ser óbvio e natural para os programadores. Como tornar este processo óbvio mas ao mesmo tempo suficientemente informativo para que estes possam ter confiança no que estão a fazer?

- As sugestões dadas pela plataforma devem ter um risco associado muito baixo para o programador, entendendo-se como risco a introdução de optimizações que causem um comportamento incorrecto das aplicações. Como poderá a nossa solução diminuir o risco associado a cada sugestão de optimização?

- Para implementarmos a nossa solução será necessário efectuar alterações à DSL. Como poderemos introduzir estas alterações de modo a que a simplicidade e a dimensão da DSL não sejam comprometidas?

- A plataforma *OutSystems* é um sistema maduro, com cerca de dez anos de existência e mais de um milhão de linhas de código. Desenvolver e validar a nossa solução em cima desta plataforma também é em si um desafio importante.

1.4 Contribuições

As contribuições obtidas com esta dissertação foram as seguintes:

1. Concebemos e desenvolvemos uma solução que permite às plataformas de desenvolvimento e suporte à execução de aplicações web, sugerir e aplicar optimizações às aplicações baseado na análise de dados estatísticos recolhidos durante a execução das aplicações. As optimizações inseridas correspondem à introdução de mecanismos de cache sobre consultas exclusivamente sobre entidades identificadas para optimização.

2. Extendemos o sistema de *profiling* existente na plataforma *Outsystems*, de modo a obter toda a informação necessária à decisão sobre que entidades devem ser mantidas em cache.

3. Desenvolvemos um serviço que através de técnicas de amostragem recolhe da base de dados informação sobre a utilização de cada entidade das aplicações.

4. Desenvolvemos uma função heurística, que com base nos dados recolhidos pelos sistemas de recolha de informação identifica os pontos das aplicações que devem ser optimizados para melhorar o desempenho dessas aplicações.

5. Dotámos o compilador da linguagem com capacidades para introduzir optimizações às aplicações através de mecanismos de cache sobre as consultas à base de dados, tanto nas entidades identificadas pela função heurística para serem optimizadas automaticamente, como nas sugeridas e aceites pelo programador.

6. Desenvolvemos um mecanismo de invalidação dos dados em cache com dependências para a base de dados, utilizado tanto nas optimizações introduzidas pela nossa solução como nos elementos mantidos em cache através dos mecanismos já existentes na plataforma.

7. Desenvolvemos um mecanismo de sugestões, que com base nos pontos identificados pela função heurística, apresenta aos programadores sugestões de melhoramentos às aplicações que efectivamente aumentam o desempenho das mesmas, introduzindo um risco associado muito baixo para o programador.

8. Criámos uma ligação entre os ambientes de desenvolvimento e de produção que possibilita ao mecanismo de sugestões apresentar aos programadores a informação relevante para suportar a decisão de aceitar ou rejeitar as sugestões apresentadas.

9. Implementámos e validámos a nossa solução sobre a plataforma *OutSystems*, onde realizámos uma avaliação do impacto causado no desempenho das aplicações que evidenciaram uma diminuição de 60% sobre a quantidade média de pedidos à base de dados.

1.5 Estrutura do Documento

Esta secção limita-se a apresentar de forma geral os vários tópicos apresentados ao longo do documento.

No segundo capítulo é apresentada a plataforma *OutSystems*, utilizada para a implementação e validação da nossa solução. Neste capítulo apresentamos o ambiente de desenvolvimento disponibilizado pela plataforma, a sua arquitectura, o ciclo de vida das aplicações com esta desenvolvidas e as fases percorridas pelos pedidos realizados pelos utilizadores finais às aplicações. Também são apresentados os sistemas de cache e de *profiling* existentes na plataforma no momento em que iniciámos o nosso trabalho.

No terceiro capítulo são apresentados alguns conceitos e abordagens úteis para o contexto deste trabalho. Em particular, é discutida uma solução geral que permite obter informação sobre a execução das aplicações e decidir quais os pontos críticos dessas aplicações, de modo a guiar o compilador a focar-se em optimizações nesses pontos. Neste capítulo são também apresentados dois tipos de optimizações estudados numa fase inicial, *inlining* e cache, tendo sido seleccionada a optimização através da introdução de mecanismos de cache por considerarmos, com base no estudo realizado, que os ganhos envolvidos nesta optimização seriam mais óbvios.

No quarto capítulo é apresentada a solução concebida para atingir os objectivos propostos e superar os desafios apresentados, seguindo-se, no quinto capítulo, a descrição da implementação da nossa solução sobre a plataforma *OutSystems*, tal como a análise dos resultados obtidos pela solução quando aplicada à aplição *IssuesManager*, apresentada na secção 1.2.2.

Por último, no capítulo 6, são apresentadas algumas conclusões e propostas de temas para trabalhos futuros.

2

OutSystems Agile Platform

Para testar o nosso trabalho utilizámos a *Agile Platform*, desenvolvida pela empresa *OutSystems*, que consiste numa plataforma de desenvolvimento e modelação de aplicações web empresariais feitas à medida. Neste capítulo vamos apresentar esta plataforma para que possamos conhecer melhor o ambiente onde a nossa solução será implementada e testada.

A *Agile Platform* disponibiliza aos programadores um modo rápido e incremental de criação e manutenção de toda a aplicação web. Para suportar o desenvolvimento de aplicações web, esta plataforma possui uma ferramenta denominada por *Service Studio*, que será apresentada na próxima secção.

2.1 *Service Studio*

O *Service Studio* é a ferramenta de suporte ao desenvolvimento de aplicações web disponibilizada pela *Agile Platform*. Esta ferramenta possibilita aos utilizadores a criação de toda a aplicação, incluindo as páginas web, a lógica da aplicação, o modelo de dados e as configurações de segurança, tudo isto num só ambiente de desenvolvimento.

A linguagem de programação que os utilizadores encontram no *Service Studio* é uma linguagem graficamente orientada. Assim, todos os elementos da aplicação são criados através de simples acções de *"Drag and Drop"* sobre elementos já existentes, configurando apenas algumas das suas propriedades.

As aplicações criadas nesta ferramenta têm como base ficheiros OML (*OutSystems Markup Language*), onde é mantida toda a informação da aplicação desenvolvida, com excepção do código de referências externas. Estes ficheiros são normalmente denominados de *eSpaces*.

Outra funcionalidade disponibilizada pelo *Service Studio* é o seu mecanismo interno de junção, que permite aos programadores modificarem em paralelo um *eSpace*, sendo posteri-

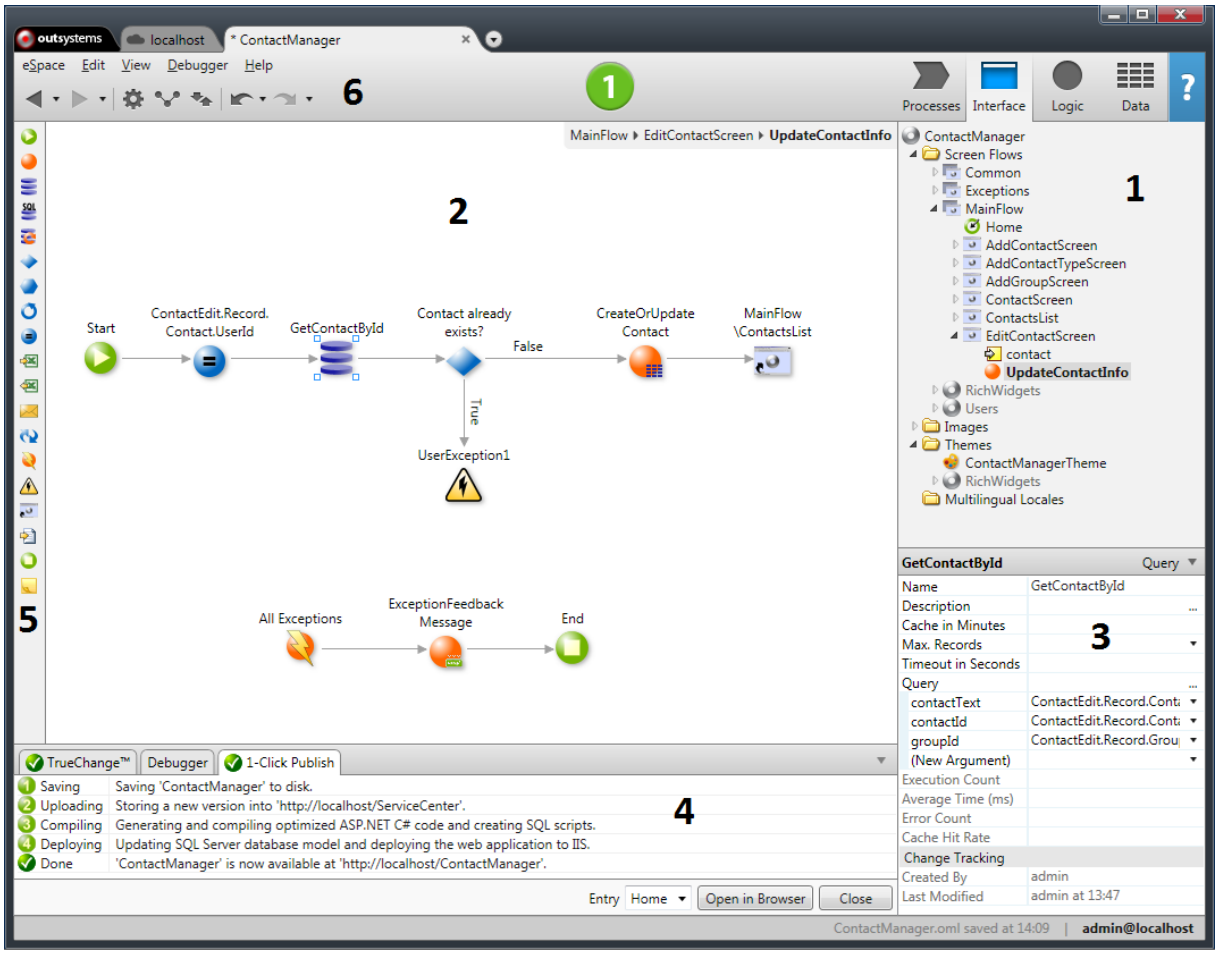

Figura 2.1: *Service Studio* - o ambiente de desenvolvimento de aplicações *OutSystems*.

ormente condensadas as várias versões num só ficheiro. Esta ferramenta permite também a verificação de consistência do modelo, procurando sobretudo erros de tipificação e de sintaxe.

A figura 2.1 apresenta o ambiente encontrado no *Service Studio*, donde vamos destacar os seguintes componentes:

Árvore de eSpace (1) - Mostra todos os elementos disponíveis no *eSpace*.

Tela de Fluxo (2) - Painel onde o programador desenha o ecrã e os fluxos de acções.

Painel de Propriedades (3) - Permite ao programador visualizar e definir propriedades sobre o elemento seleccionado, tanto pela *Tela de Fluxo* como pela *Árvore de eSpace*.

Painel Inferior (4) - Apresenta ao programador erros e avisos referentes ao *eSpace* e permite também observar o comportamento de execução da aplicação quando esta se encontra em modo de *debug*. O estado do processo de compilação e publicação de um *eSpace* pode também ser seguido pelo programador através de mensagens apresentada neste painel.

Árvore de Ferramentas (5) - Contém os vários elementos que podem ser adicionados aos fluxos de acções. Para adicionar estes elementos a um fluxo de acções basta utilizar o tradicional método de *"Drag and Drop"*.

Barra de Ferramentas (6) - Para além dos habituais comandos, como abrir ou gravar um ficheiro, disponibiliza também comandos para compilar e publicar a aplicação, executar a aplicação, visualizar o diagrama de entidades e relações, entre outros comandos úteis ao programador.

2.1.1 Linguagem de Programação Visual

Uma linguagem para domínios específicos (DSL) é uma linguagem de programação ou de especificação dedicada a um determinado tipo de problemas ou de modelação de problemas.

O *Service Studio* implementa uma DSL visual que permite ao programador criar aplicações web recorrendo apenas a construções de alto nível, disponibilizando aos programadores uma forma rápida de definir os fluxos de acções pretendidos (*Action Flows*).

Os principais elementos de alto nível desta linguagem são os *web Flows*, que definem as ligações entre as várias páginas da aplicação, os *web Screens* e os *web Blocks*, que definem graficamente a interface visual da aplicação, os *Action Flows*, que definem partes da lógica das aplicações e as *Entities*, que definem o modelo de dados.

Um *web Flow* é representado por um grafo orientado, contendo apenas um nó inicial e um conjunto ilimitado de outros nós. Cada um destes nós representa um *web Screen* e as arestas representam as transições possíveis entre os vários *web Screens*. Cada *web Screen* correponde a uma página da aplicação.

Um *Action Flow* é também representado por um grafo orientado, contendo apenas um nó inicial e um conjunto de um ou mais nós terminais, onde cada nó do grafo corresponde a uma operação da linguagem, e cada aresta define a próxima operação a executar.

Para tratar eventos gerados pela interacção do utilizador com a aplicação, a linguagem permite a criação de procedimentos. Estes procedimentos são denominados por *Actions*. Uma *Action* pode ter um conjunto de parâmetros de entrada e de saída, podendo também definir variáveis locais a esse procedimento, e tem o seu próprio *Action Flow*.

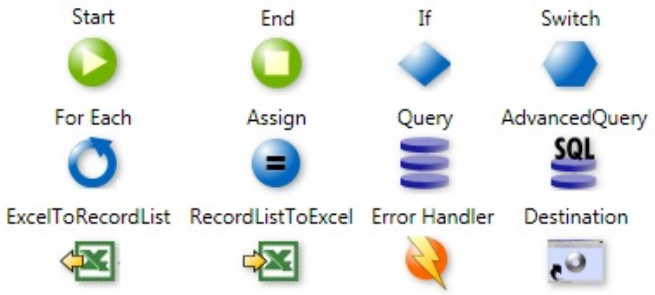

Figura 2.2: Algumas construções da linguagem *OutSystems*.

De seguida são apresentadas as principais construções da linguagem integrada no *Service Studio* (figura 2.2):

Start & End - Delimitam um *Action Flow*.

Assign - Permite atribuir valores a variáveis.

If & Switch - Controlam o fluxo de execução através da avaliação de expressões.

Foreach - Repete a execução de um caminho do fluxo para cada entrada da lista associada a esta construção.

Query - Executa uma consulta à base de dados, selecionando informação sobre uma ou mais entidades declaradas no *eSpace*. Esta construção tem associada uma interface gráfica que permite ao programador definir os elementos da consulta sem ter de inserir código manualmente (figura 2.3).

Advanced Query - Executa uma consulta à base de dados sobre uma ou mais entidades declaradas no *eSpace*, sendo que nesta construção é o próprio utilizador que define manualmente o código da consulta a executar (figura 2.3). A linguagem usada na criação da consulta é uma linguagem semelhante ao standard SQL.

Destination - Desvia o fluxo de execução de um *Action Flow* para uma página web criada pela aplicação (*web Screen*).

Error Handler - Permite capturar uma excepção e executar uma rotina de tratamento.

Excel to RecordList - Importa o conteúdo de um ficheiro do tipo MS-Excel.

Record List To Excel - Exporta o conteúdo de uma lista de registos para um ficheiro do tipo MS-Excel.

Execute Action - Executa qualquer acção dísponível para o *eSpace*. Dos vários tipos de acções disponíveis destacam-se as *Entity Actions*, *User Actions*, *Referenced Actions* e *Built-In Actions*.

Entity Actions - Conjunto de acções criadas automaticamente para cada entidade do modelo. Estas acções são criadas no momento da definição de uma entidade e consistem em operações sobre a base de dados para inserção, remoção e edição de elementos nessa entidade.

User Actions - Conjunto de acções definidas pelo programador. Estas acções têm a particularidade de poderem ser reutilizadas por outros *eSpaces* caso assim se deseje.

Referenced Actions - Acções externas ao *eSpace* importadas para uso local.

Built-In Actions - Conjunto de acções de sistema disponíveis para os programadores.

A figura 2.4 apresenta um exemplo de um *Action Flow* para adicionar um contacto à base de dados, verificando se o seu email é válido e se este já existe na base de dados.

Tal como foi referido anteriormente, o *Service Studio* permite-nos também construir a camada de visualização da aplicação. O ambiente de desenvolvimento desta camada da aplicação está apresentado na figura 2.5.

Depois de terem sido desenvolvidas, as aplicações estão prontas para serem compiladas e publicadas no *Platform Server*, o sistema de suporte à execução para aplicações *OutSystems*.

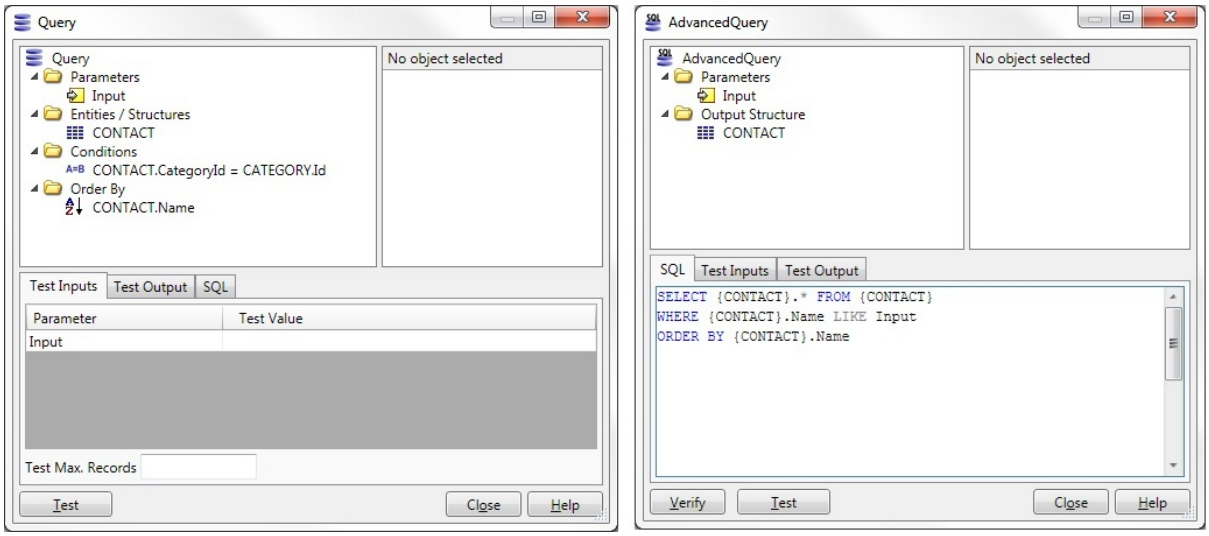

Figura 2.3: Painel para edição de *Queries* e *Advanced Queries*.

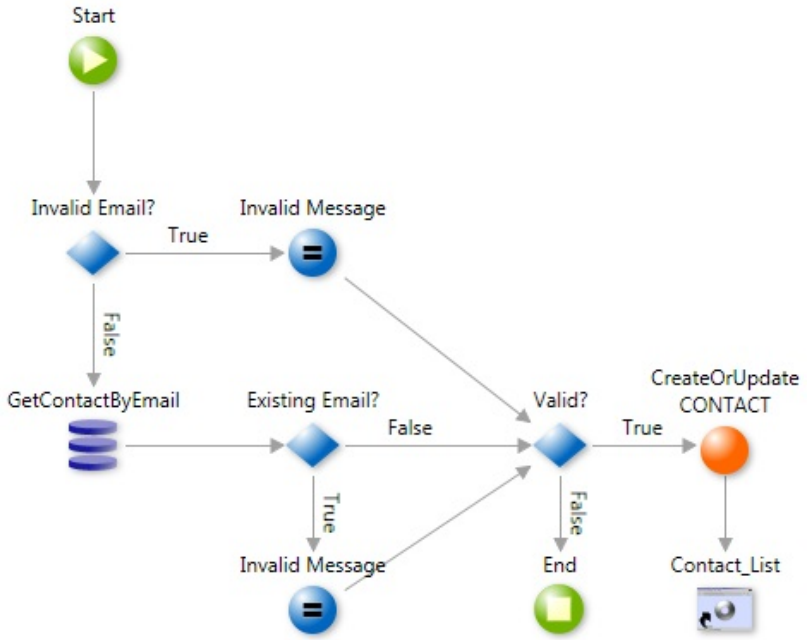

Figura 2.4: Exemplo de um *Action Flow* para adicionar um contacto a uma base de dados.

2.2 *Platform Server*

O *Platform Server* é a plataforma de suporte à execução de aplicações web *OutSystems*. A arquitectura desta plataforma está apresentada na figura 2.6, sendo composta pelos seguintes componentes principais:

Front-End Server - Um *Front-End Server* corresponde a um típico servidor de aplicações web com alguns serviços adicionais:

- *Service Center*: consola de administração do *Platform Server*.

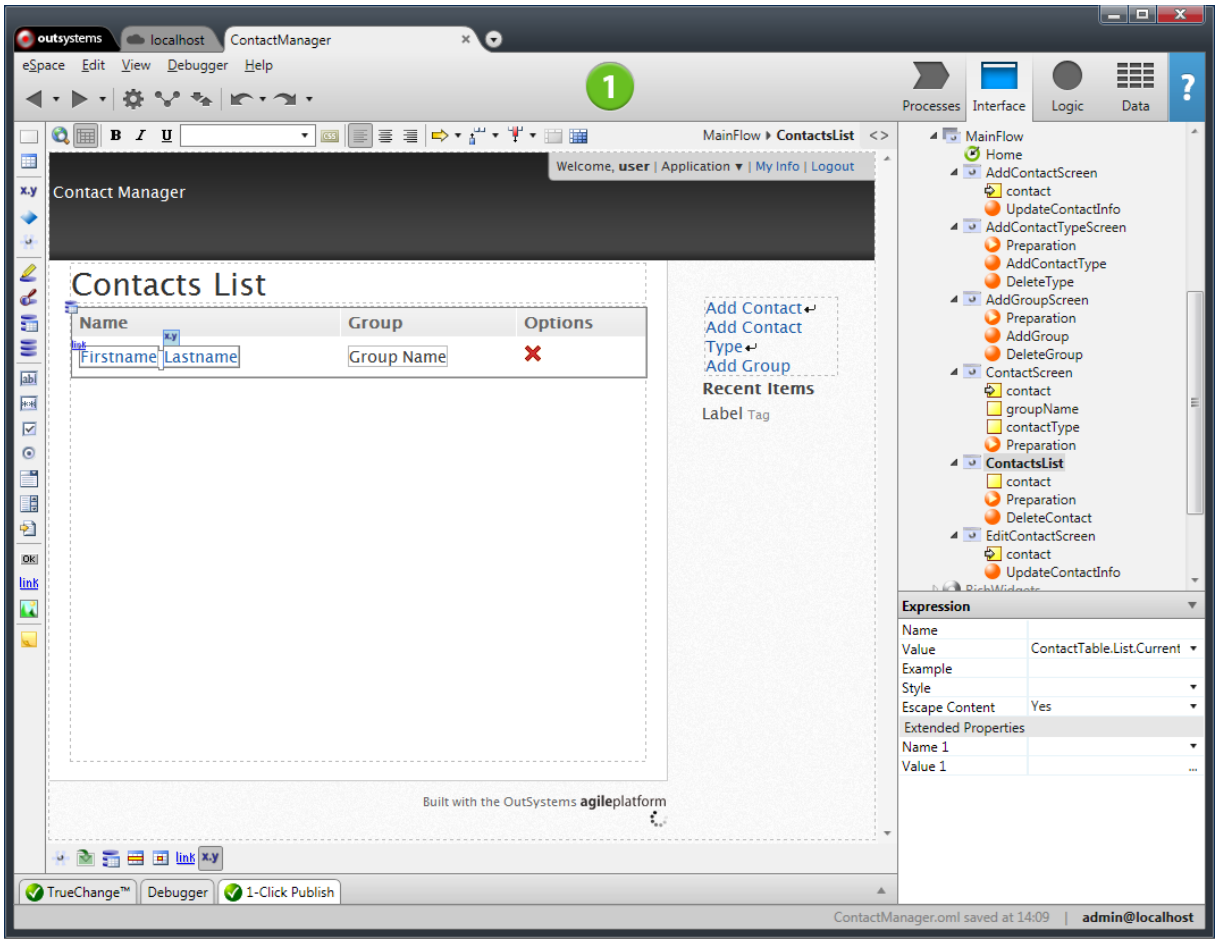

Figura 2.5: Ecrã encontrado no *Service Studio* para edição da camada de visualização.

- *Deployment Service*: serviço que em conjunto com o *Deployment Controller Service*, garante que as aplicações depois de compiladas são instaladas nos servidores de aplicações web.
- *Log Service*: serviço que assíncronamente regista os erros gerados pelas aplicações em execução.

Deployment Controller Server - Responsável pela compilação do código das aplicações web e posterior publicação dos ficheiros resultantes da compilação nos *Front-End Servers*.

Database Server - Sistema de gestão de bases de dados relacionais, como o Microsoft SQL Server ou o Oracle.

Esta secção e a anterior permitiram-nos conhecer as componentes disponibilizadas pela *Agile Platform* para suportar o desenvolvimento e execução das aplicações web *OutSystems*. A secção seguinte apresenta o modo como estas se ligam entre si no processo de desenvolvimento e publicação de aplicações web.

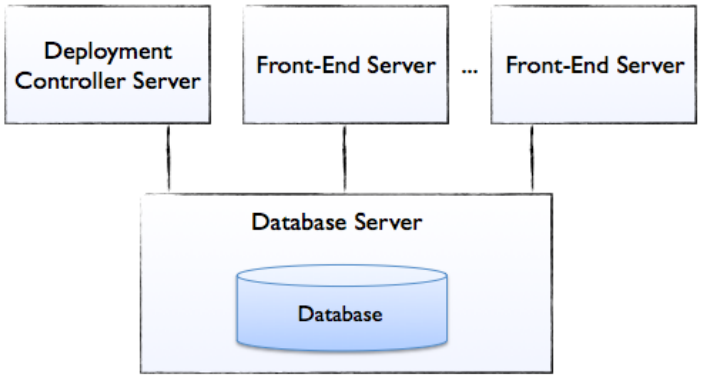

Figura 2.6: Arquitectura do *Platform Server*

2.3 Ciclo de Vida das Aplicações *OutSystems*

Quando analisamos o ciclo de vida das aplicações *OutSystems* encontramos dois ambientes distintos: o ambiente de desenvolvimento, onde os programadores desenham, implementam e testam as suas aplicações, e o ambiente de produção, onde a aplicação é publicada de modo a tornar-se acessível aos utilizadores finais.

Considerando o ciclo de vida das aplicações apresentado na figura 2.7, podemos descrever esse ciclo pelos seguintes passos principais:

1. Os programadores utilizam o *Service Studio* para desenhar e implementar as suas aplicações web. Sempre que for necessário testar as aplicações, estas podem ser publicadas para o ambiente de desenvolvimento.

2. Após as aplicações terem sido submetidas a uma fase de testes e caso seja tomada a decisão de publicar essas aplicações para o ambiente de produção, o *delivery manager* é responsável por tal transição.

3. Os utilizadores finais utilizam as aplicações publicadas para o ambiente de produção.

4. O *delivery manager* recebe por parte dos utilizadores *feedback* sobre as suas experiências de utilização.

5. O *feedback* dado pelos utilizadores finais é analisado pelo *delivery manager* e passado aos programadores para que estes possam realizar optimizações às aplicações.

Vamos agora apresentar com um pouco mais de detalhe a operação de publicação de um *eSpace*.

2.3.1 Processo de Publicação

Após concluído o desenho do *eSpace*, podemos executar a operação de publicação do *eSpace* para o ambiente de desenvolvimento. Esta operação pode ser facilmente executada clicando sobre o *icon* de *1Click-Publish* (1CP), existente na barra de ferramentas do *Service Studio*.

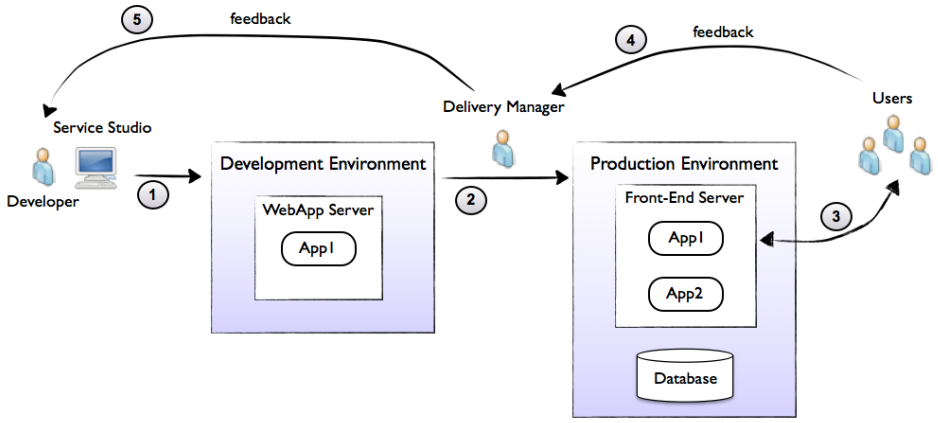

Figura 2.7: Ciclo de vida das aplicações *OutSystems*.

Esta operação engloba quatro passos principais:

Save - Guarda o *eSpace* no computador utilizado para desenvolvimento.

Upload - Faz o *upload* do *eSpace* (OML) para o *Platform Server* ao qual se está conectado.

Compile - Este passo é executado no *Platform Server* ao qual se está conectado e envolve a tradução do ficheiro OML, criado pelo *Service Studio*, num conjunto de ficheiros Microsoft .NET ou Java.

Deploy - Último passo executado pela operação. Actualiza a versão do *eSpace* publicado.

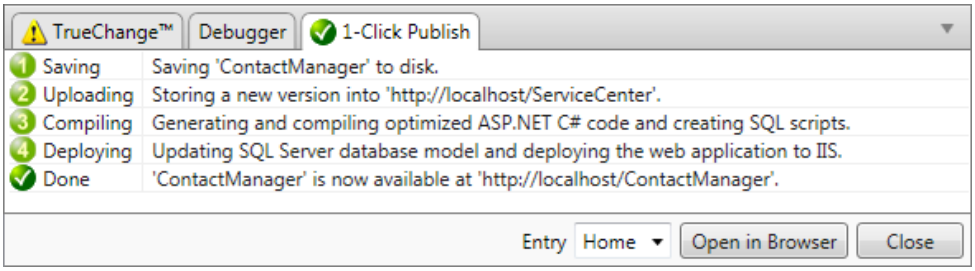

Figura 2.8: Painel inferior do *Service Studio* mostrando o progresso da operação *1-Click Publish*.

A progressão da operação de *1-Click Publish* pode ser seguida pelo programador através do painel inferior do *Service Studio* (Figura 2.8). Neste painel são também colocados erros e avisos, caso tenham sido encontrados problemas no *eSpace* durante a operação de *1CP*.

Esta operação permite ao programador abstrair-se da implementação dos serviços necessários à compilação e alocação da aplicação, pois com um só clique são desencadeados todos os serviços necessários a este processo.

A figura 2.9 apresenta as várias operações desencadeadas quando o programador solicita a operação de *1-Click Publish*.

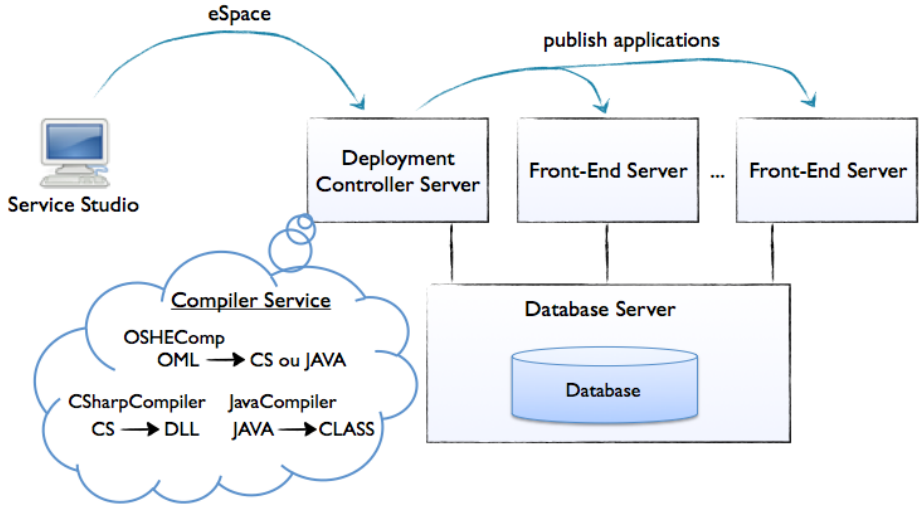

Figura 2.9: Visão geral da operação *1-Click Publish*.

Analisemos agora com mais detalhe o processo de compilação e publicação de um *eSpace*. Tal como foi mencionado anteriormente, a compilação de um *eSpace* é efectuada no *Platform Server*. Esta operação pode ser dividida em duas fases principais. A primeira fase é responsável por traduzir o ficheiro OML em ficheiros C# através do compilador da linguagem (OSHEComp). Dentro deste processo podemos identificar os seguintes passos:

Load - Carrega o código existente no ficheiro OML.

Analyse/Optimize - Analisa o código carregado e efectua possíveis optimizações, com o objectivo de melhorar o desempenho da aplicação final.

Dump C# - Cria os ficheiros C# correspondentes ao código OML carregado, tendo em consideração as optimizações introduzidas.

Dump Proxies - Cria os ficheiros C# correspondentes a proxies necessários pela aplicação.

Após concluída esta primeira fase de compilação foi gerado todo o código C# necessário e pode-se iniciar a segunda fase deste processo. Nesta fase recorre-se ao compilador da linguagem C# (CSC) para traduzir o código C# obtido no passo anterior em ficheiros .DLL.

Concluída a fase de compilação de código, os vários ficheiros .DLL gerados são guardados no *Platform Server* na directoria *share*. Esta directoria contém uma sub-directoria para cada aplicação publicada no servidor.

Aquando do processo de publicação de um *eSpace*, os .DLLs correspondentes a esse *eSpace* guardados na directoria *share*, são publicados nos vários *Front-End Servers* de modo a tornar a aplicação acessível na web. Na realidade não são publicados apenas os ficheiros .DLL correspondentes ao *eSpace* mas também os .DLLs doutros *eSpaces* que sejam referenciados por este. Este processo encontra-se ilustrado na figura 2.10.

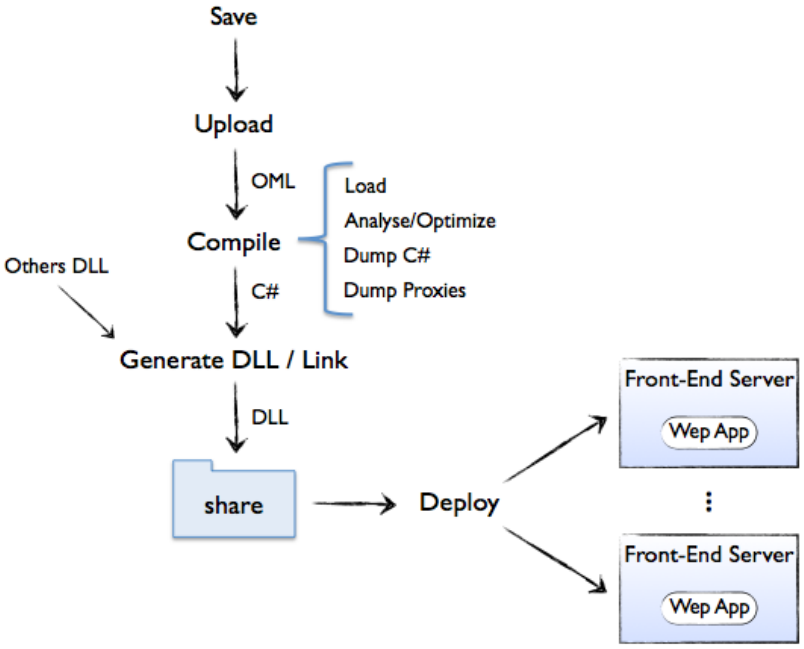

Figura 2.10: Visão detalhada da operação *1-Click Publish*.

Depois de termos apresentado tanto o processo de desenvolvimento como o de publicação de aplicações web, na secção seguinte vamos analisar o ciclo de vida dos pedidos realizados sobre aplicações web em produção.

2.4 Pedidos nas Aplicações *OutSystems*

Após efectuada a operação de *1-CP* as aplicações estão prontas a serem utilizadas pelos utilizadores finais. Neste capítulo vamos descrever o ciclo de vida de um pedido iniciado por um utilizador, através do seu *browser*, a uma aplicação *OutSystems* em produção.

O processo desencadeado quando um utilizador, através do seu *browser*, invoca um pedido sobre uma aplicação *OutSystems*, está representado na figura 2.11.

Como podemos observar, o *browser* mantém duas variáveis com informação sobre o *View State* e o *SessionID*. O *View State* é um atributo escondido existente nas páginas web, que contém a informação necessária para persistir a mudanças de estado de formulários web. O *SessionID*, tal como o nome indica, consiste num valor que permite identificar a sessão do utilizador. No *browser* é também mantida uma cache onde são guardados ficheiros estáticos, permitindo um acesso mais rápido a estes.

Quando é solicitado um pedido a uma aplicação *OutSystems*, o *browser* constrói o respectivo pedido HTTP, onde é colocado o valor das variáveis *View State* e *SessionID*, e encaminha esse pedido pela rede até ao servidor de aplicações web (IIS - Internet Information Services).

O servidor de aplicações web é composto pelos seguintes componentes principais:

Application Domain - local onde são tratados os pedidos e produzidas as respostas. Contém

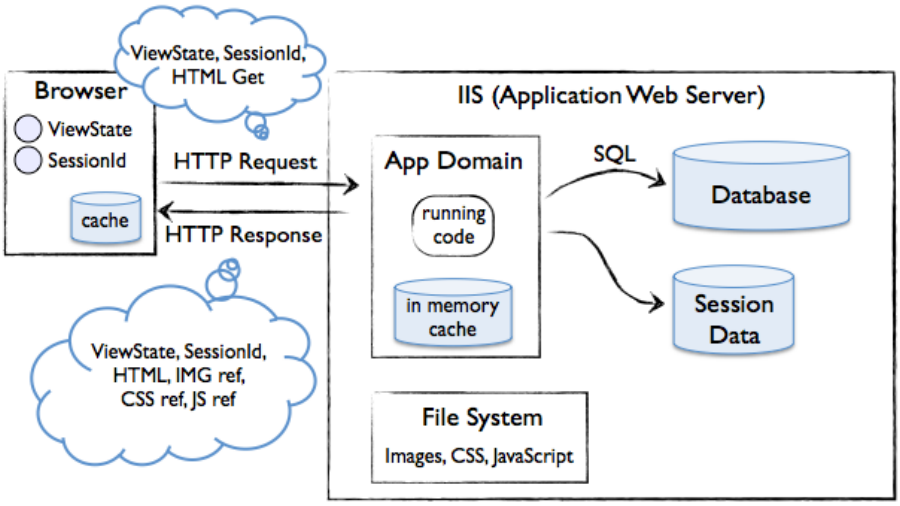

Figura 2.11: Ciclo de vida de um pedido sobre uma aplicação *OutSystems*.

um sistema de cache de ficheiros estáticos para diminuir o tempo de resposta aos pedidos.

Application Data Repository - repositório onde estão guardados os dados necessários às aplicações. O método de acesso a estes dados é através de queries SQL.

Session Data Repository - repositório onde são guardados os dados das sessões.

File System - sistema de ficheiros onde se encontram guardados os ficheiros estáticos necessários à execução das aplicações, como por exemplo ficheiros de imagens, css e javascript.

Após o tratamento dos pedidos e criação da respectiva resposta, o servidor de aplicações web envia ao utilizador uma resposta HTTP. Nessa resposta, para além do código HTML da página pretendida, é enviado também o *View State*, o *SessionID* e referências para os ficheiros estáticos necessários à apresentação correcta da página. Depois de receber essa resposta, o *browser* cria um novo pedido HTTP por cada referência a ficheiros estáticos existente na resposta inicial que não possa ser satisfeita pela sua cache. Por último o *browser* apresenta ao utilizador a resposta obtida.

Como podemos observar, desde o momento em que o utilizador envia um pedido ao *browser* por uma determinada página, até ao momento em que essa página lhe é apresentada, existem vários componentes intervenientes, cada uma com o seu impacto no tempo de espera do utilizador.

Na próxima secção é apresentado um sistema existente na plataforma, que permite recolher informação relevante sobre a execução das aplicações.

2.5 Sistema de *Profiling OutSystems*

No âmbito dum projecto anterior [1], desenvolveu-se na *OutSystems* um sistema de *profiling* cuja função é recolher dados estatísticos em tempo de execução sobre a utilização das aplicações

web desenvolvidas com a plataforma *OutSystems*.

O principal objectivo deste sistema é ajudar os programadores a detectar pontos de congestão das aplicações, permitindo assim ao programador realizar optimizações sem que se atinjam situações críticas a nível do desempenho dessas aplicações. Os dados recolhidos podem também ser usados pelos programadores para medir o impacto causado pela alteração de código nas aplicações existentes.

Para satisfazer o objectivo atrás referido, o sistema de *profiling* utiliza um mecanismo baseado em instrumentação de código para recolher, para cada elemento da linguagem, informação sobre: o tempo de execução, o número de vezes que executou com sucesso, o número de vezes que a execução terminou com erro e o número médio de iterações de cada ciclo *for-each*. O impacto deste sistema no desempenho das aplicações foi considerado como aceitável não introduzindo um grande custo relativamente aos possíveis ganhos deste sistema.

A informação recolhida pelo sistema é posteriormente apresentada ao programador através do *Painel de Propriedades* do *Service Studio*, como se pode observar na figura 2.12.

Este sistema permite ajudar os programadores a detectar pontos de congestão das aplicações, para que estes possam optimizar esses mesmos pontos. Uma optimização muitas vezes considerada quando se pretende melhorar o desempenho de aplicações web consiste na utilização de *cache*. A próxima secção apresenta o mecanismo de *cache* existente na *Agile Platform*.

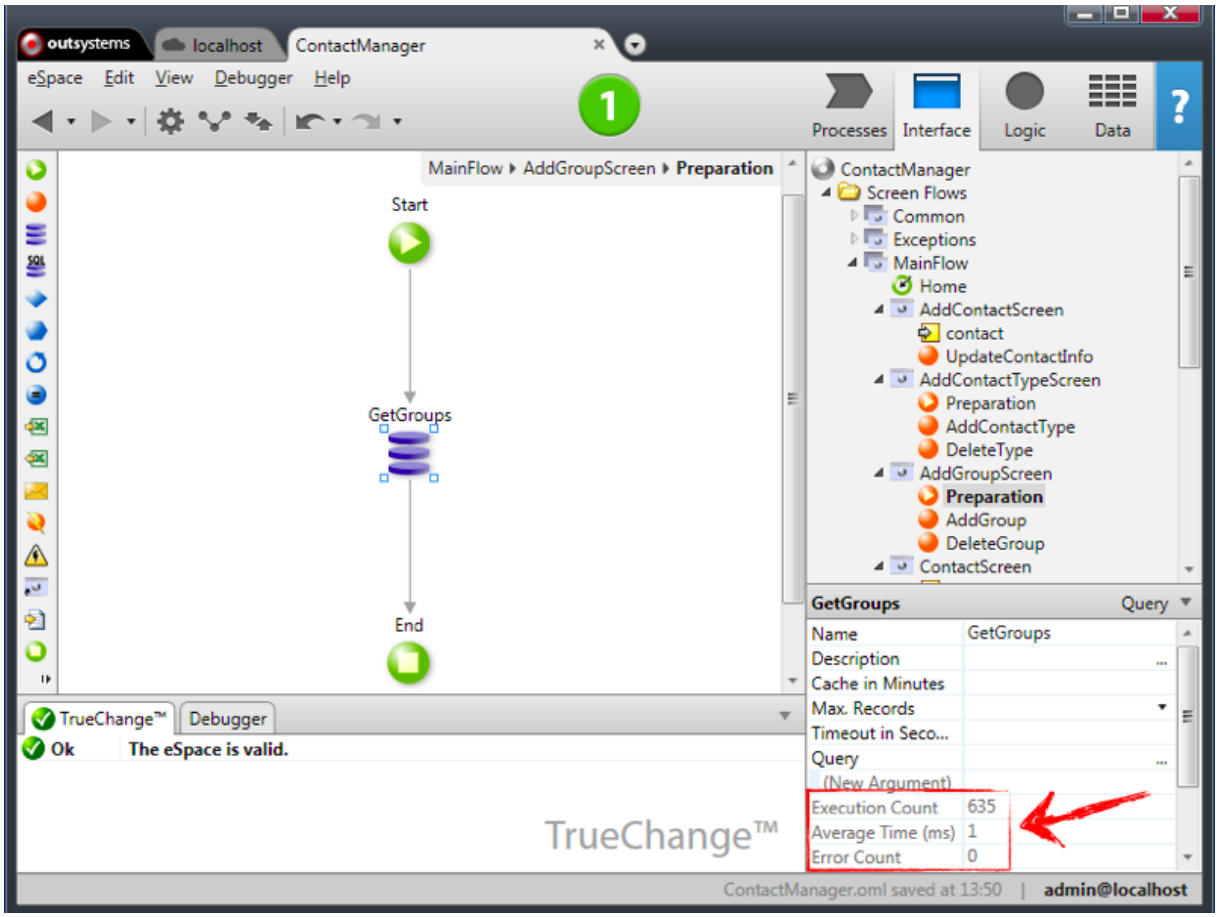

Figura 2.12: *Service Studio* com informação estatística recolhida pelo sistema de *profiling*.

2.6 Cache em *OutSystems*

A plataforma *OutSystems* disponibiliza aos seus programadores um mecanismo de cache para tentar melhorar o desempenho das aplicações desenvolvidas. Esse mecanismo de cache é denominado por *CacheInMinutes* e é realizado ao nível do servidor aplicacional, seguindo um mecanismo *Read-Through*, com invalidação por tempo ou explícita, através de acções existentes na plataforma. No âmbito de um trabalho anterior [39], foi estudado e implementado um mecanismo de cache distribuída, tendo os resultados finais evidenciado que o custo de comunicações associado a este mecanismo não seria benéfico para o desempenho das aplicações comparativamente com o sistema de cache existente.

2.6.1 *CacheInMinutes*

O *CacheInMinutes* é um mecanismo de cache que permite ao programador, para cada elemento onde é possível ser aplicada, indicar durante quando tempo (definido em minutos) se pretende manter esse elemento em cache até que este seja invalidado e removido da mesma. Esta informação pode ser introduzida através do *Painel de Propriedades* de cada elemento. Os elementos em que o programador pode recorrer ao uso de cache são os seguintes: *Web Screens*, *Web Blocks*, *Actions*, *Queries* e *Advanced Queries*.

Este mecanismo de cache foi desenvolvido recorrendo às classes *Cache* e *CacheStore* disponibilizadas pelas plataformas ASP.NET e JAVA respectivamente, que seguem um mecanismo de cache com entradas do tipo (chave,valor). A cada elemento guardado em cache corresponde uma chave única, composta pelo identificador do *eSpace* ao qual esse elemento corresponde, o identificador do utilizador da aplicação, e pelo identificador do objecto guardado. Cada elemento guardado em cache contém duas dependências para ficheiros, uma para um ficheiro que permite identificar o *eSpace* correspondente e outra para um ficheiro que permite identificar o utilizador da aplicação. O uso destas dependências é explicado na secção seguinte.

2.6.2 Invalidação da Cache

Tal como foi referido anteriormente, quando o programador pretende utilizar o mecanismo de cache disponibilizado pela plataforma *OutSystems*, este indica, para cada elemento pretendido, durante quantos minutos se pretende manter esse elemento em cache até que este seja invalidado e removido. Este corresponde a um dos três mecanismos de invalidação existentes na plataforma. Os outros dois mecanismos de invalidação fazem uso dos ficheiros de dependências associados a cada elemento em cache e são accionados através do uso de duas acções pré-definidas e disponibilizadas no *ServiceStudio*, a *TenantInvalidateCache* e a *EspaceInvalidateCache*. Estas acções permitem ao programador invalidar todos os elementos guardados em cache que pertençam a um determinado utilizador ou *eSpace*, respectivamente. A execução destas acções consiste apenas em modificar o respectivo ficheiro de dependências para que a cache seja invalidada.

2.7 Discussão

Neste capítulo foram apresentados os principais componentes da *Agile Platform*, a plataforma de desenvolvimento e modelação de aplicações web empresariais desenvolvida pela empresa *OutSystems*. Começámos por apresentar o ambiente de desenvolvimento disponibilizado pela *Agile Platform*, o *Service Studio*. De seguinda apresentámos a plataforma de suporte à execução de aplicações web, o *Platform Server*, e o ciclo de vida das aplicações *OutSystems* e dos pedidos realizados sobre estas. Por último apresentámos o sistema de *profiling* e o mecanismo de cache existentes na *Agile Platform*.

Este estudo foi essencial para o nosso projecto pois será este o ambiente que servirá de exemplo para testar a nossa solução e analisar os resultados obtidos.

3

Conceitos e Abordagens Relacionadas

Neste capítulo são apresentados alguns conceitos e abordagens relacionadas com o tipo de solução que pretendemos implementar, cujos objectivos são permitir a recolha de informação sobre a execução das aplicações e a identificação e optimização de pontos críticos. Neste capítulo são apresentados os três grandes componentes existentes neste tipo de solução: sistema de *profiling*, sistema de tomada de decisão e optimizador.

De seguida é apresentada uma contextualização que nos permite identificar a importância de recorrer a uma abordagem deste tipo para a optimização de aplicações web.

3.1 Contexto

Um problema directamente relacionado com o desenvolvimento de aplicações web é o desempenho das mesmas durante a interacção com os utilizadores finais. Existem vários factores que podem diminuir o desempenho das aplicações web [12, 31], como por exemplo a latência na rede, largura de banda, sobrecarga dos servidores, tempo necessário à criação das páginas nos servidores, tempo de comunicação com servidores de bases de dados e quantidade de dados transportados pela rede. Alguns destes factores podem ser optimizados, como por exemplo o tempo gasto pelos servidores na construção de páginas ou a quantidade de dados transportados pela rede, enquanto que outros dependem completamente de factores externos, como por exemplo a latência na rede, a largura de banda e o tempo necessário para comunicar com os servidores de bases de dados. Contudo, apesar da optimização destes últimos factores não estar nas mãos dos programadores, as aplicações podem ser optimizadas de modo a contornar estes problemas. Por exemplo, o tempo associado aos acessos a bases de dados é composto pelo tempo de comunicação com os servidores, que por sua vez é condicionado pela latência na rede e pela largura de banda, e pelo tempo de obtenção dos dados. Apesar de um programador não

ter controlo sobre a latência na rede nem sobre a largura de banda, se este conseguir diminuir o número de acessos a bases de dados, diminui também o impacto que estes factores têm no desempenho das aplicações.

Para contornar os problemas atrás referidos podem ser inseridas optimizações de dois modos. Estas optimizações podem ser introduzidas pelos programadores explicitamente no código das aplicações, ou podem ser introduzidas optimizações automaticamente no momento de compilação dessas aplicações, sem que seja necessária a intervenção do programador. Quando nos encontramos no segundo caso, em que o compilador é dotado de capacidades para melhorar o desempenho das aplicações, é necessário ter em conta vários factores:

Dimensão da aplicação - o tempo necessário ao compilador para realizar optimizações está dependente da quantidade de dados que este tem de analisar;

Impacto no desempenho - o compilador apenas deve realizar optimizações que de facto melhorem o desempenho das aplicações, pois uma optimização que seja útil num determinado cenário pode ser prejudicial noutro.

Para resolver estes problemas têm sido realizados vários estudos nos últimos anos no sentido de dotar o compilador com informação sobre quais as sequências de acções de cada aplicação que são realizadas com maior frequência, para que o compilador se possa focar apenas na optimização dessas sequências de acções. Este processo é denominado por optimizações guiadas por dados estatísticos e encontra-se apresentado na secção seguinte.

3.2 Optimizações Guiadas por Dados Estatísticos

Quando se pretende dotar um compilador com capacidades para melhorar automaticamente as aplicações é necessário que este consiga inferir quais os pontos das aplicações que necessitam de ser optimizados e que optimizações devem ser efectuadas em cada ponto. Esta necessidade torna-se mais clara se pensarmos que as aplicações podem ser bastantes grandes e complexas, o que levaria o compilador a gastar muito tempo em optimizações caso este não estivesse informado à partida sobre que pontos necessitariam realmente de ser optimizados.

Para resolver estes problemas, têm sido realizados vários estudos nos últimos anos, no sentido de dotar o compilador de informação sobre quais as sequências de acções que são realizadas com maior frequência em cada aplicação (*"hot spots"*) [4]. Sabendo quais são esses pontos, o compilador pode então focar-se neles, não desperdiçando tempo em optimizações que não influenciem significativamente o desempenho final das aplicações [22, 42, 43].

Para a identificação e optimização de pontos críticos, pode ser concebida uma solução, baseada na apresentada em [4], constituída pelos seguintes componentes principais:

- **sistema de *profiling***, para recolher informação durante a execução das aplicações que permita identificar os pontos candidatos para possíveis optimizações;

- **sistema de tomada de decisão**, que através da análise dos dados recolhidos pelo sistema

de *profiling* decida quais os pontos candidatos a optimizações e que optimizações devem ser aplicadas a cada ponto;

- **optimizador**, para efectuar em tempo de compilação as optimizações nos pontos seleccionados pelo sistema de tomada de decisão.

Todos estes componentes devem introduzir um custo mínimo de modo a melhorar tanto o desempenho final da aplicação como o tempo necessário à compilação da mesma. As figuras 3.1 e 3.2, apresentam o processo de detecção de pontos críticos numa plataforma sem sistema de *profiling* e noutra com esse sistema implementado.

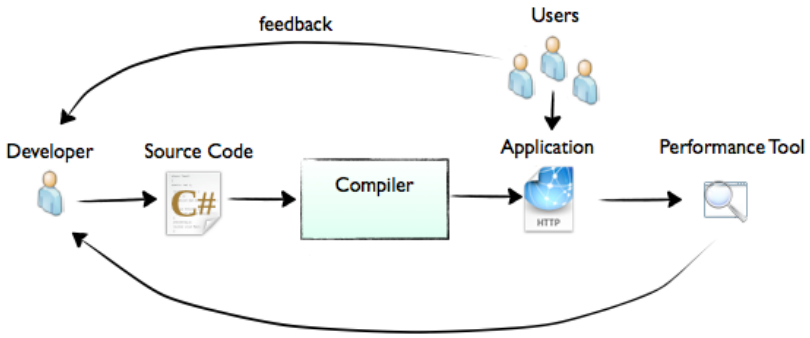

Figura 3.1: Processo de detecção de pontos críticos numa plataforma sem um sistema de *profiling* implementado.

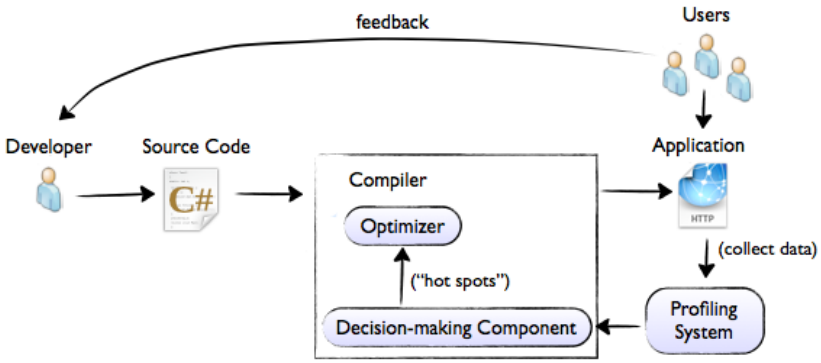

Figura 3.2: Processo de detecção de pontos críticos numa plataforma com um sistema de *profiling* implementado.

3.3 Sistema de *Profiling*

O sistema de *profiling* é responsável por recolher informação suficiente durante a execução das aplicações, para que seja possível ao sistema de tomada de decisão identificar os pontos candidatos à introdução optimizações. Dois mecanismos de *profiling* usados para recolha de informação são: amostragem (*sampling*) [7, 41] e instrumentação de código [10, 14, 16].

3.3.1 Amostragem

O mecanismo baseado em amostragem permite recolher um subconjunto dos eventos gerados pela execução da aplicação. Restringindo o sistema de amostragens para observar apenas uma percentagem dos eventos totais restringe-se também o custo associado a este sistema. Um exemplo deste tipo de sistema é usado nos ambientes de máquinas virtuais, onde periodicamente é analisada a pilha de invocações de métodos para recolher informação sobre quais os métodos mais utilizados. A SELF-93 [19] e a Japaleño VM [5] são dois exemplos de sistemas que utilizam esta aproximação para poderem focar as suas optimizações nos métodos executados com maior frequência.

3.3.2 Instrumentação de Código

Outro mecanismo de *profiling* é a instrumentação de código. Este mecanismo consiste em adicionar automaticamente código às aplicações que permita recolher um grande conjunto de dados com uma granularidade fina. Exemplos de dados que se podem recolher com este mecanismo são o número de vezes que um determinado método executou e o tempo médio de execução. Muitos estudos foram realizados usando em tempo de compilação dados recolhidos pelo sistema de *profiling* através de instrumentação de código para guiar o compilador no momento de introdução de optimizações nas aplicações. Apesar de alguns destes estudos produzirem resultados positivos no desempenho das aplicações [8], outros demonstraram que o custo associado à instrumentação era bastante elevado, prejudicando o desempenho das mesmas [9].

A principal causa destas divergências nos resultados deve-se à diferença nos tipos de instrumentação utilizada, donde concluímos que apesar de ser possível obter um grande conjunto de dados com esta técnica, é necessário usá-la com precaução pois esta pode tornar-se bastante prejudicial para as aplicações, como se pode observar em [9] onde a instrumentação de código tornou o desempenho das aplicações, durante a recolha de dados, cerca de cem vezes mais lento.

3.4 Sistema de Tomada de Decisão

Com base nos dados obtidos pelo sistema de *profiling*, a função desta componente é identificar os pontos candidatos a sofrerem optimizações e decidir que optimizações deverão ser aplicadas a cada um desses pontos [4].

Um método para decidir que pontos são considerados como candidatos consiste em seguir uma simples estratégia baseada em limites [4]. Com esta estratégia os dados recolhidos para cada ponto são analisados e submetidos a uma heurística, que irá produzir um valor final, que deverá ser um valor indicativo da importância desse ponto no desempenho da aplicação. Caso esse valor ultrapasse o limite estabelecido pela componente, esse ponto será escolhido para ser optimizado. A escolha da função heurística deve também ter em consideração o tipo de aplicação que se pretende optimizar.

Analisemos agora um pequeno exemplo. Consideremos uma pequena aplicação de partilha de músicas onde os utilizadores têm ao seu dispor funcionalidades como: visualizar informação sobre músicas, álbuns, artistas e outros utilizadores, entre outras. Para isso, a aplicação utiliza um modelo de dados com várias entidades como por exemplo as entidades músicas, estilos musicais, artistas, álbuns e utilizadores. Suponhamos que se pretende optimizar esta aplicação através da utilização de mecanismos de cache automáticos, e que o sistema de *profiling* existente recolhe informação sobre o número de vezes que cada entidade foi acedida para leitura e para escrita. Neste caso, uma boa função heurística seria estabelecer uma relação entre o número de acessos de leitura e o número de acessos de escrita, de modo a favorecer as entidades que são frequentemente acedidas para leitura e que sofrem poucas operações de escrita. Voltando ao nosso exemplo, a função heurística deveria seleccionar por exemplo a entidade *estilos musicais* para ser optimizada visto que os seus dados variam pouco e são acedidos muito frequentemente, mas não deveria seleccionar a entidade *músicas* pois esta é actualizada com alguma frequência.

3.5 Optimizador

Após terem sido identificados pelo sistema de tomada de decisão os pontos que devem ser optimizados e quais as optimizações que devem ser efectuadas, estão reunidas as condições para que o optimizador realize a sua tarefa: efectuar as optimizações à aplicação. Existem várias optimizações que podem ser realizadas, desde optimizações de mais baixo nível, como é o caso da realocação de registos [33], a optimizações de mais alto nível, como é o caso do *inlining* [6, 17, 38] ou mecanismos de cache [13, 20, 32].

De seguida apresentamos as optimização de *inlining* e cache.

3.5.1 *Inlining*

Invocações de métodos podem tornar-se em barreiras para algumas optimizações, pois o código que será executado com essas invocações não é local ao método que se encontra a ser optimizado. O objectivo do *inlining* é remover não só estas barreiras, como também o custo associado à invocação de métodos [21]. Para isso, uma invocação sobre um método é substituída por uma cópia do corpo desse método. A figura 3.3 apresenta um exemplo de um método antes e depois da optimização de *inlining*.

Esta optimização pode não só introduzir melhorias no desempenho das aplicações elimi-

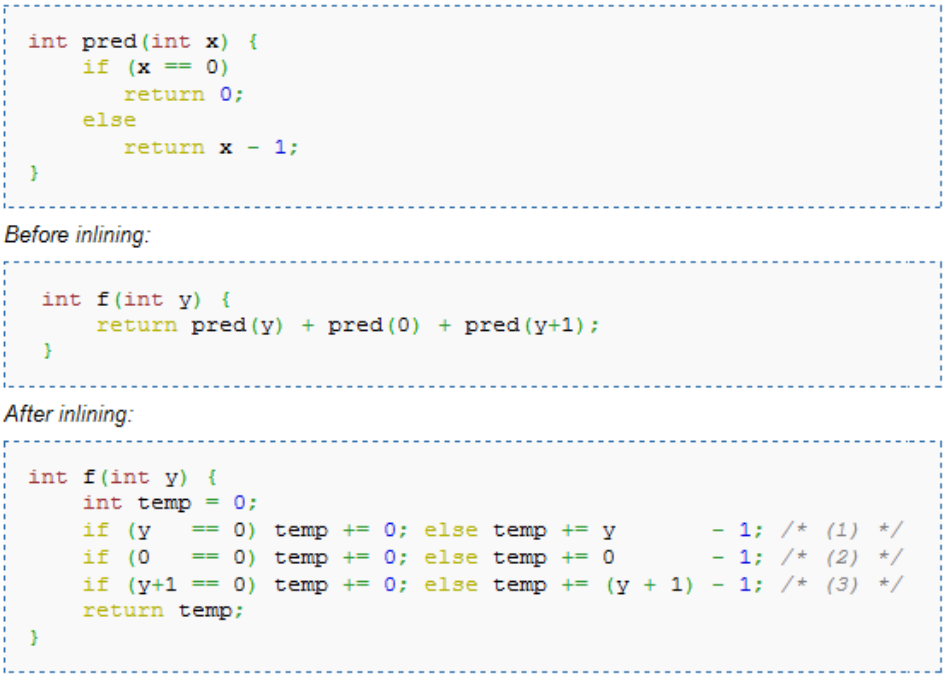

Figura 3.3: Resultado da aplicação da optimização de *inlining* a um método.

nando o tempo inerente ao tratamento de invocações de métodos, como também torna possível a outras optimizações serem mais eficientes, pois passam a conhecer exactamente o código que seria executado pela invocação ao método.

Tal como qualquer outro tipo de optimização, o uso incorrecto do *inlining* pode prejudicar o desempenho das aplicações. O principal problema desta optimização diz respeito à dimensão do código da aplicação. Um exemplo de um problema que pode surgir com o aumento da dimensão do código é a possibilidade de um método crescer de tal modo que não seja possível fazer uso da cache de instruções, o que poderá levar a uma perda de desempenho na execução da aplicação. Outro problema desta optimização deve-se ao facto de alguns sistemas, como é o caso da JVM, estabelecerem um limite máximo à dimensão dos métodos [23], o que pode levar a que algumas expansões não possam ser efectuadas. Contudo, estes problemas podem ser evitados caso o sistema de tomada de decisão, apresentado na secção 3.4, apenas seleccione um conjunto de pontos suficientemente pequeno para que o código não cresça em demasia, mas suficientemente rico no que diz respeito às melhorias que esta optimização pode trazer no desempenho desses pontos e consequentemente no desempenho da aplicação em geral.

Este tipo de optimizações é utilizado em vários sistemas, onde se obtêm resultados positivos aplicando a optimização apenas aos pontos críticos das aplicações. Exemplos de sistemas que utilizam esta aproximação são o compilador de SELF-93 [19] e a Japaleño VM [5].

3.5.2 Cache

Uma *cache* é um mecanismo conhecido para melhorar o desempenho dos sistemas e consiste em realizar um armazenamento de objectos de dados recentemente usados, num local mais

próximo de quem os irá solicitar [12]. Quando um cliente requisita um objecto, o serviço de cache primeiro verifica se possui uma cópia desse objecto e caso isso aconteça o mesmo é entregue ao processo cliente. Caso o objecto não esteja armazenado na *cache* o mesmo é acedido directamente na sua origem.

Ao analisarmos os sistemas onde são usados mecanismos de cache encontramos dois tipos de abordagens que são seguidas. A primeira é a que encontramos quando analisamos sistemas de computadores e consiste em manter os dados na mesma localização geográfica mas guardando-os em dispositivos de memória com maior velocidade de acesso e de transmissão de dados [37]. Este tipo de abordagem é encontrada por exemplo nos computadores pessoais, onde são implementados vários níveis de cache (CPU, RAM, disco) [36]. A segunda abordagem é utilizada em sistemas web, onde clientes e servidores comunicam entre si para satisfazerem os pedidos realizados. Neste tipo de sistemas, a abordagem de cache utilizada consiste em alterar a localização dos dados de modo a colocá-los mais próximos de quem os irá solicitar, permitindo assim um acesso mais rápido aos mesmos através da diminuição do tempo associado às comunicações na rede [12, 20].

Vamos agora analisar, no contexto específico de aplicações web, os vários pontos onde é possível utilizar-se mecanismos de cache.

A figura 3.4 apresenta uma visão geral das componentes existentes em ambientes de aplicações web, tal como as comunicações entre elas. Neste diagrama podemos identificar pelo menos três pontos onde se recorre ao uso de mecanismos de cache: Browser, Servidor de Aplicações e Servidor de Bases de Dados.

Nesta secção vamos centrar-nos apenas em mecanismos de cache ao nível do Servidor de Aplicações, pois é sobre este nível que o nosso trabalho se irá focar.

Quando se desenvolve um mecanismo de cache é necessário tomar decisões sobre como se devem processar os acessos de leitura e de escrita sobre elementos guardados em cache. A próxima secção apresenta alguns mecanismos de cache que podem ser adoptados para definir o modo como esses acessos são processados.

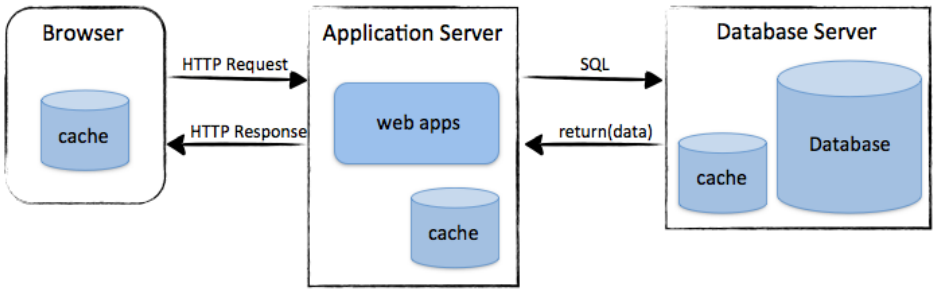

Figura 3.4: Visão geral das componentes e comunicações entre elas num ambiente típico de aplicações web.

3.5.3 Estratégias de Cache

Quando se implementa um mecanismo de cache é necessário ter em consideração que tipo de abordagem será usada no que diz respeito a acessos de leitura e escrita sobre dados guardados na cache. De seguida são apresentadas algumas estratégias que podem ser utilizadas por estes mecanismos [34].

Read-Through

Num mecanismo de cache com esta estratégia de leitura, quando uma aplicação realiza um pedido sobre a cache por uma determinada entrada, por exemplo pela chave K, se a chave K não se encontrar em cache é realizado o procedimento necessário para obter o respectivo valor e carregá-lo para a cache. Caso essa entrada já exista na cache o seu valor é simplesmente obtido e devolvido.

Write-Through

Num mecanismo de cache com esta estratégia de escrita, quando uma aplicação realiza uma actualização sobre dados guardados em cache, esta não é concluída até que essas alterações sejam efectuadas no respectivo repositório de dados. Com esta estratégia, o desempenho de operações de escrita é penalizado com a existência do mecanismo de cache, pois para cada escrita é necessário realizar-se uma operação de escrita sobre os dados em cache e efectuar-se uma ligação ao repositório de dados, sendo esta operação realizada de modo síncrono.

Write-Behind

Esta estratégia é uma alternativa à anterior e tem como objectivo melhorar o desempenho das escritas sobre dados em cache. Nesta estratégia, as escritas sobre a cache não são imediatamente efectuadas no repositório de dados. Em vez disso, a cache guarda informação sobre que entradas foram alteradas, sendo estas alterações posteriormente efectuadas sobre o repositório de dados de modo assíncrono, penalizando ligeiramente a coerência dos dados relativamente à estratégia write-through.

Refresh-Ahead

Esta estratégia consiste em recarregar para a cache, de forma automática e assíncrona, dados que tenham sido acedidos recentemente e que estejam prestes a expirar. O resultado desta estratégia é que caso uma entrada acedida muito frequentemente esteja guardada na cache, a aplicação não irá sentir o impacto do recarregamento dos dados do repositório para a cache quando essa entrada expirar o seu tempo. Esta actualização da cache de forma assíncrona apenas é efectuada quando uma entrada que está prestes a expirar é acedida. Caso uma entrada seja acedida após expirar o seu tempo de cache, será realizada uma leitura de forma síncrona dos dados guardados no repositório, sendo estes posteriormente mapeados para a cache.

Como a plataforma que será usada para validar e testar a nossa solução produz código ASP.NET, a próxima secção apresenta o mecanismo de cache disponibilizado neste contexto.

3.5.4 Cache em .NET

Quando se fala de cache ao nível do servidor de aplicações podemos encontrar pelo menos quatro tipos de elementos passíveis de serem guardados em cache. Podemos fazer cache de páginas ou partes de páginas da aplicação, de ficheiros estáticos (imagens, css, etc) e cache de dados. Neste trabalho vamos focar-nos apenas em cache de dados, pois pretende-se melhorar o desempenho das aplicações reduzindo o número de acessos a bases de dados através do uso de cache dos dados guardados nas bases de dados.

O ASP.NET possui um mecanismo de cache que permite às aplicações guardar objectos em memória permitindo um acesso mais rápido aos mesmos. Este mecanismo de cache é implementado pela classe *Cache*, que disponibiliza ao programador uma interface simples de usar. Cada instância desta classe é local a uma aplicação, sendo o seu tempo de vida definido pelo tempo de vida da aplicação associada. Assim, quando uma aplicação é reiniciada o objecto *Cache* é recriado, perdendo-se a informação nele guardada.

Classe *Cache*

A classe *Cache* usa o típico padrão de pares chave-valor no método de inserção de dados, podendo estes serem obtidos posteriormente através da chave com que foram inseridos.

Para adicionar elementos à cache esta classe disponibiliza aos programadores dois métodos, o *Insert* e o *Add*. As principais diferenças entre estes dois métodos dizem respeito ao retorno dos métodos e ao comportamento quando se insere um elemento cuja chave já existe em cache. Enquanto que o método *Insert* tem como tipo de retorno *void*, o método *Add* retorna o objecto inserido na cache. Relativamente ao comportamento na inserção de elementos com chaves já existentes em cache, enquanto que o método *Insert* substitui o elemento existente em cache pelo novo elemento, o método *Add* não o faz, mantendo em cache o elemento já guardado ignorando assim o novo elemento. A classe *Cache* disponibiliza várias implementações destes métodos que permitem ao programador associar ao elemento inserido políticas de invalidação e dependências.

Para obtenção de objectos guardados no objecto *Cache* é disponibilizado o método *Get*, que permite obter um elemento guardado na cache através da chave associada a esse elemento. Caso não exista nenhum elemento na cache com uma chave que corresponda à pretendida o método devolve o valor *null*.

Para além da classe *Cache* disponibilizar uma interface simples que permite ao programador adicionar e obter elementos, esta oferece também funcionalidades que permitem ao programador configurar como os elementos são guardados e durante quanto tempo irão persistir em cache. Uma funcionalidade inerente a um mecanismo de cache diz respeito ao processo de invalidação de dados quando se atinge a dimensão máxima de ocupação disponível para a cache e se pretende adicionar novos elementos a esta.

Invalidação da Cache

Para ajudar o mecanismo de cache na selecção de elementos para serem removidos da cache o programador pode associar a cada elemento uma prioridade. Adicionando esta propriedade aos elementos estamos a informar a cache sobre a importância de cada um deles, para que no processo de invalidação sejam removidos os elementos com nível de prioridade mais baixa. Os níveis de prioridade disponíveis estão definidos no enumerado *CacheItemPriority* e são os seguintes (por ordem crescente no nível de prioridade): *Low*, *BellowNormal*, *Normal* ou *Default*, *AboveNormal*, *High* e *NotRemovable*. A associação de um nível de prioridade a um elemento é realizada no momento de inserção desse elemento em cache.

No momento de inserção de um elemento em cache o programador pode também estabelecer uma política de expiração para esse elemento. Para especificar o tempo de vida de um elemento podemos utilizar o parâmetro *absoluteExpiration*, do tipo *DateTime*, que permite especificar o momento temporal exacto em que um elemento deve expirar. Outra propriedade que se pode definir é a quantidade de tempo, começando no momento do último acesso, que um elemento pode estar em cache sem ser acedido. Para isso pode-se utilizar o parâmetro *slidingExpiration*, do tipo *TimeSpan*. Sempre que um elemento expira é invalidado e removido da cache, sendo retornado o valor *null* caso se tente obter esse valor, a menos que este seja novamente adicionado à cache.

Para além das funcionalidades atrás apresentadas, o ASP.NET permite definir a validade de um elemento guardado em cache baseado num ficheiro externo, numa directoria, noutro elemento guardado em cache ou em dados guardados em bases de dados. Estas funcionalidades são denominadas por dependências de ficheiros e dependências de chaves. Caso uma dependência sofra alterações o elemento guardado em cache é invalidado e removido da cache. Esta técnica pode ser usada para remover elementos da cache quando as suas fontes de dados se alteram. Por exemplo, consideremos uma aplicação cujo objectivo é produzir um grafo a partir de dados processados de um ficheiro XML. Esses dados podem ser guardados em cache colocando uma dependência para o ficheiro XML para que os dados em cache sejam invalidados e removidos caso o ficheiro sofra alterações. Assim é possível evitar inconsistências entre os dados utilizados na produção do grafo e os dados guardados no ficheiro fonte.

O ASP.NET suporta também um mecanismo de notificações que permite ao programador definir métodos a executar quando um elemento está prestes a expirar ou após a sua remoção da cache.

Uma parte significativa das aplicações web empresariais têm os seus dados guardados em servidores de bases de dados. Para que se possam guardar esses dados em cache com garantia que não irão surgir inconsistências entre os dados guardados em cache e os dados guardados nas bases de dados, podem ser definidas dependências entre estas duas componentes.

Dependências entre Cache e Bases de Dados

Para se definirem dependências entre os elementos guardados em cache e tabelas ou linhas de tabelas guardadas em bases de dados, o ASP.NET disponibiliza a classe *SqlCacheDependency*.

Com este tipo de dependências, sempre que ocorre uma alteração numa tabela ou numa linha específica de uma tabela, as entradas guardadas em cache associadas a essa tabela ou linha são invalidadas e removidas da cache. Esta classe permite definir dependências sobre tabelas em bases de dados Microsoft SQL Server 7.0, SQL Server 2000, SQL Server 2005 e SQL Server 2008. Para bases de dados SQL Server 2005 e SQL Server 2008 é possível definir dependências sobre linhas específicas duma tabela. Em sistemas SQL Server 7.0 e 2000, este mecanismo é implementado seguindo um modelo de *polling*, existindo uma *thread* na aplicação ASP.NET que periodicamente verifica se os dados foram alterados na base de dados. Em sistemas SQL Server 2005 e 2008 este modelo foi melhorado, implementando agora um modelo baseado em notificações. Com este modelo, o servidor de bases de dados é responsável por enviar notificações às aplicações sempre que dados guardados sejam modificados.

O uso de mecanismos de cache pode aumentar drasticamente o desempenho das aplicações em determinados cenários mas pode também provocar inconsistências entre os dados guardados em cache e nos ficheiros fonte. Por exemplo, consideremos uma aplicação do tipo *e-commerce*, que apresenta informação sobre produtos guardados numa base de dados. Sem um mecanismo de cache, esta aplicação terá de efectuar um pedido à base de dados sempre que um utilizador pretenda visualizar informação sobre um produto. Se essa informação for guardada em cache com uma política de expiração temporal de um dia, podemos garantir respostas mais rápidas por parte do servidor a pedidos de clientes sobre essa informação, pois essa informação já se encontra guardada em memória não sendo assim necessário aceder à base de dados. Contudo, caso a informação sobre esse produto seja alterada na base de dados, a informação guardada na cache poderá encontrar-se inconsistente relativamente à nova informação durante um período máximo de um dia (tempo de expiração definido no momento de inserção da informação na cache), podendo o utilizador ser induzido em erro devido a inconsistências de dados.

Usando cache com dependências para a base de dados podemos guardar a informação sobre o produto em cache e criar uma dependência sobre uma tabela ou linha na base de dados onde essa informação se encontra guardada. Assim, caso esses dados sejam alterados na base de dados, a informação guardada na cache é invalidada e removida. Da próxima vez que um utilizador solicitar essa informação, se essa informação não estiver em cache, é obtida da base de dados e adicionada à cache, garantindo assim que se obtém sempre a última versão dos dados diminuindo assim os problemas de inconsistência dos dados.

Este mecanismo pode ser também utilizado em *Web Farms* (ambientes com múltiplos servidores), sendo todos eles informados das notificações.

Um pequeno exemplo da utilização deste mecanismo encontra-se apresentado na listagem 3.1. Neste exemplo é inserido na cache um elemento com a chave *"Employees"* associado à informação sobre todas as entradas existentes na tabela *Employees* (resultado do comando definido na linha 4 do exemplo). O terceiro parâmetro passado no método *Insert* (linha 15) cria uma dependência entre o elemento a inserir na cache e os dados existentes na tabela *Employees* da base de dados. Assim, caso esses dados sofram alterações esta entrada será invalidada e removida da cache.

Para bases de dados Oracle, o ASP.NET disponibiliza a classe *OracleCacheDependency* que é em tudo semelhante à classe *SqlCacheDependency* mencionada anteriormente.

Um outro modo de obter este comportamento em bases de dados Oracle, consiste em definir dependências entre os dados em cache e ficheiros criados no servidor, sendo criado um ficheiro para cada dependência diferente. De seguida pode-se criar um procedimento no servidor Oracle, podendo este ser escrito na linguagem C ou em JAVA, para modificar o ficheiro utilizado na dependência. Por último pode-se criar um *trigger* para que esse procedimento seja executado sempre que haja uma alteração sobre determinados dados na base de dados. O *trigger*, ao executar o procedimento que modifica o ficheiro, invalida os elementos em cache que dependem desse mesmo ficheiro, tornando assim os dados guardados em cache consistentes com os da base de dados.

Listing 3.1: Inserção de um elemento em cache com dependências para a base de dados.

```
1  string connectionString =
2     webConfigurationManager.ConnectionStrings["databaseName"].ConnectionString;
3  SqlConnection connection = new SqlConnection(connectionString);
4  string tQuery = "SELECT EmployeeID, FirstName, LastName, City FROM dbo.Employees";
5  SqlCommand command = new SqlCommand(tQuery,connection);
6  SqlDataApdater adapter = new SqlDataAdapter(command);
7
8  DataSet employeesDataset = new DataSet();
9  adapter.Fill(employeesDataset, "Employees");
10
11 // Cache Dependecy
12 SqlCacheDependency empDependency = new SqlCacheDependency(command);
13
14 // Add a cache item that will be invalidated if one of its records changes
15 Cache.Insert("Employees", employeesDataSet, empDependency);
```

3.6 Discussão

Neste capítulo apresentou-se uma possível abordagem para a concepção da solução que se pretende obter com este trabalho. Começámos por apresentar a importância que este tipo de solução pode ter em ambientes de desenvolvimento de aplicações web, seguindo-se uma apresentação dos componentes principais dessa solução: o *Sistema de Profiling*, responsável por recolher informação estatística sobre a execução das aplicações, o *Sistema de Tomada de Decisão*, responsável por analisar a informação recolhida e identificar os pontos críticos das aplicações, e o *Optimizador*, responsável por efectuar as optimizações nos pontos identificados como sendo críticos.

Uma solução seguindo esta abordagem permite aos compiladores optimizarem automaticamente as aplicações focando-se apenas nos pontos críticos das mesmas.

4

Desenho da Solução

Neste capítulo é apresentado o desenho de uma solução cujo objectivo é introduzir nas plataformas de desenvolvimento e suporte à execução de aplicações web capacidade de sugerir optimizações e optimizar as aplicações com estas desenvolvidas, sendo este processo guiado pela análise de dados reais de utilização das aplicações recolhidos durante a sua execução.

Na secção 1.2.1 apresentámos alguns problemas inerentes ao processo de identificação e optimização dos pontos críticos das aplicações levado a cabo pelos programadores. Um problema inerente ao processo de identificação de pontos críticos deve-se à falta de informação disponibilizada aos programadores em ambiente de desenvolvimento sobre a utilização dessas aplicações em produção. Caso os programadores tivessem acesso a esta informação estes poderiam identificar facilmente os pontos críticos das aplicações o que tornaria este processo bastante eficiente.

Para tentar colmatar esta falta de informação, as aplicações são submetidas a testes de desempenho e de carga recorrendo a ferramentas que permitem simular ambientes de utilização reais. Contudo, simular um ambiente idêntico ao que será posteriormente encontrado em produção é um desafio que requer um investimento muito significativo em tempo, competências e atenção por parte das equipas de desenvolvimento, competindo assim por recursos já escassos no seio de uma organização não dedicada ao desenvolvimento web. Como consequência destas limitações, por vezes os pontos críticos das aplicações apenas são identificados quando estas, já em produção, começam a apresentar falhas a nível de desempenho. Relativamente ao processo de optimização das aplicações, para além das limitações encontradas devido aos escassos recursos atrás apresentados, optimizar as aplicações requer um conhecimento mais profundo das tecnologias envolvidas, conhecimento esse que pode levar à necessidade das equipas de desenvolvimento de recorrerem a programadores mais experientes, o que muitas vezes não é possível.

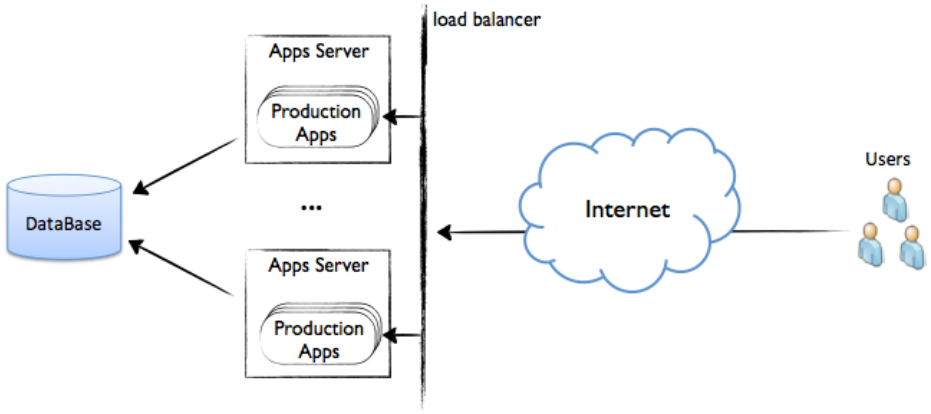

Figura 4.1: Diagrama de utilização das bases de dados pelos servidores de aplicações web.

Analisando a generalidade das aplicações web empresariais encontramos uma característica comum na maioria delas, que corresponde também a um ponto crítico muito frequente neste tipo de aplicações. Essa característica é a centralização dos seus dados em bases de dados. Este é um ponto crítico pois apesar dos pedidos sobre os dados poderem ser satisfeitos por vários servidores diferentes, a base de dados de dados utilizada pelas várias aplicações é a mesma (figura 4.1).

A solução proposta pretende ultrapassar as limitações atrás apresentadas facilitando o máximo possível ao programador o processo de identificação e optimização de pontos críticos, passando essa responsabilidade para a plataforma de suporte ao desenvolvimento e execução das aplicações.

Apesar da nossa abordagem poder ser utilizada para suportar facilmente diferentes tipos de optimizações, no contexto deste trabalho focámo-nos na optimização dos acessos a bases de dados. Para isso decidimos introduzir mecanismos de cache [11, 18, 37] sobre entidades existentes nos modelos de dados das aplicações. A gestão da cache é realizada ao nível do servidor de aplicações, onde a activação da propriedade cache sobre entidades é reflectida pela introdução de cache dos resultados de todas as consultas que utilizem apenas entidades com esta propriedade activa.

A escolha deste tipo de optimização deve-se ao facto deste ser um ponto crítico muito frequente no contexto de aplicações web empresariais, tendo esta decisão sido suportada pela análise realizada sobre uma aplicação padrão (apresentada na secção 1.2.2), donde observámos que um número significativo de entidades foram consideradas como sendo *read-mostly*.

A decisão sobre introdução de cache sobre entidades e não sobre consultas individualmente deve-se ao facto de ao detectar-se uma entidade cujos dados não sofrem alterações durante períodos de tempo, o programador apenas necessita de activar a propriedade sobre um ponto da aplicação, a entidade, em vez de ter de activar essa propriedade para todas as consultas sobre essa entidade, o que poderia tornar o processo de optimização por parte do programador pouco eficiente.

Para satisfazer estes requisitos, decidimos alterar o ciclo de vida normal das aplicações web

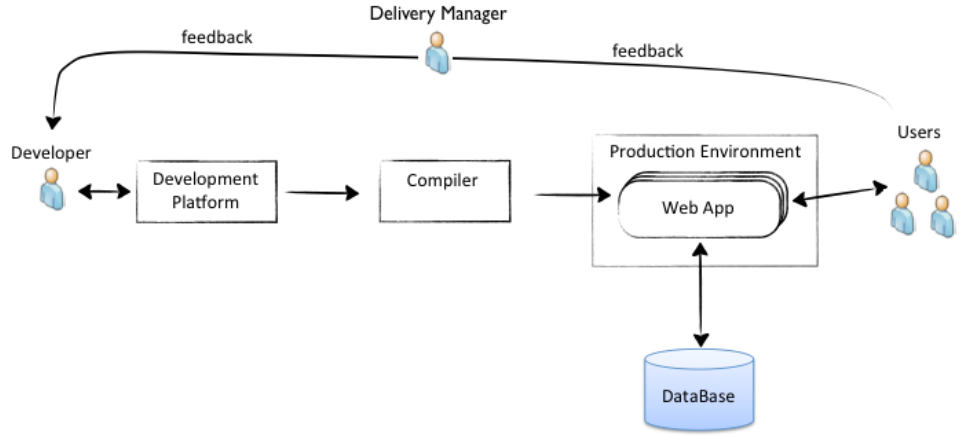

Figura 4.2: Processo típico de desenvolvimento e optimização de aplicações web.

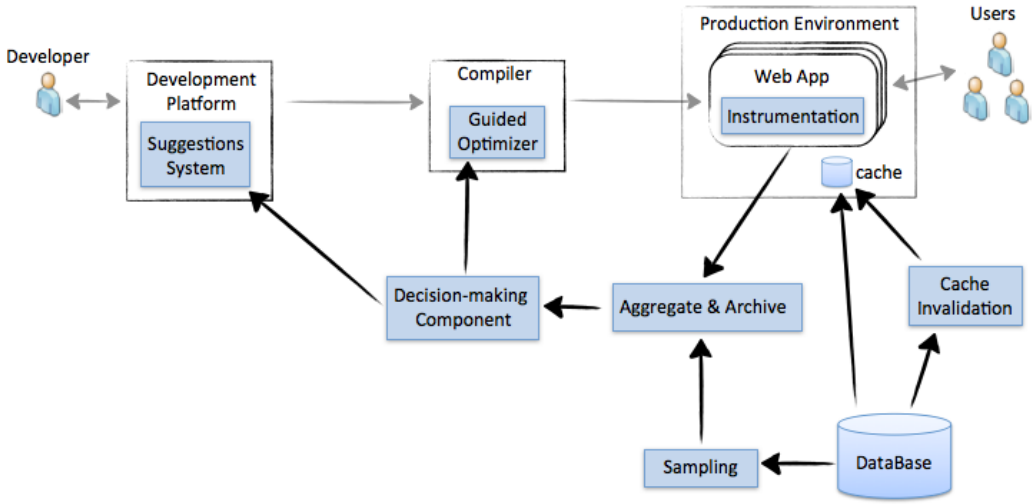

Figura 4.3: Desenho geral da solução.

empresariais (figura 4.2) para obtermos uma solução, baseada na solução apresentada em [4], constituída pelos seguintes componentes principais (figura 4.3):

Instrumentação de código - utilizando técnicas de instrumentação de código recolhe dados sobre a utilização das aplicações durante a interacção com os utilizadores.

Amostragem sobre a base de dados - através de técnicas de amostragem recolhe informação sobre a utilização de cada entidade existente nos modelos de dados das aplicações.

Agregação e arquivo de dados - agrega os dados recolhidos pelos componentes anteriores e arquiva-os de forma persistente.

Componente de Tomada de Decisões - analisa os dados arquivados pelo componente anterior e decide que entidades devem ser optimizadas automaticamente e que entidades devem ser apresentadas como sugestões de optimização.

Mecanismo de sugestões - apresenta aos programadores as sugestões de optimização e envia para o compilador a decisão do programador para cada sugestão (aceita / rejeita).

Optimizador guiado por dados de utilização - introduz cache sobre as entidades identificadas como *read-mostly*.

Invalidação de cache - responsável por invalidar as entradas em cache sempre que os dados originais sejam modificados na base de dados.

4.1 Instrumentação e Amostragem

Para determinar que optimizações devem ser sugeridas ou aplicadas directamente às aplicações é necessário obter-se informação sobre a execução dessas aplicações, sem causar um impacto significativo no tempo de execução das mesmas.

Esta componente é responsável por recolher informação sobre a utilização das aplicações em execução para que através da sua análise se possa guiar o processo de identificação de pontos críticos.

Para reduzir o impacto do processo de recolha de informação no desempenho das aplicações decidimos que este seria dividido em dois componentes:

Recolha de dados por instrumentação - através de técnicas de instrumentação de código [7] recolhe dados reais sobre a execução das aplicações durante a interacção entre estas e os utilizadores finais, permitindo por exemplo calcular a quantidade de cache *hits* e de cache *misses* para cada entidade mantida em cache.

Recolha de dados por amostragem - utilizando técnicas de amostragem [41], através da análise de tabelas de estatísticas mantidas pelos sistemas de gestão de bases de dados, recolhe dados reais sobre a utilização de cada entidade. Exemplos de informação que pode ser recolhida é a dimensão dos dados efectivamente guardados na base de dados ou a quantidade de acessos de leitura e escrita sobre cada entidade.

Esta separação de funções permite que o custo associado à recolha de informação sobre a utilização da base de dados não cause impacto no desempenho das aplicações podendo este processo ser realizado como um serviço independente.

Com a introdução destes dois componentes na nossa solução, pretendemos que se possa capturar um conjunto de informação bastante rico de modo a suportar não só o processo de identificação de pontos críticos, como também o mecanismo de sugestões, onde deve ser apresentada ao programador informação suficiente sobre a utilização das aplicações para que estes possam aceitar essas sugestões com confiança.

4.2 Agregação e Arquivo de Dados

A principal função deste componente é agregar a informação obtida pelas várias componentes de recolha de dados e arquivá-la de forma persistente. Para a agregação de dados considerámos

que devia ser agregada toda a informação obtida num determinado dia, de modo a produzir-se um histórico diário sobre a utilização de cada entidade. A justificação para se agregar informação com uma granularidade diária deve-se apenas a esta ser a unidade mais natural para caracterizar o inicio e fim de um ciclo trabalho, o que pensamos ser também a granularidade que permite ao programador uma análise mais simples dessa informação.

Tendo sido agregada a informação recolhida pelos vários componentes é necessário arquivá-la de modo a torná-la persistente para que mais tarde esta informação possa ser acedida com o intuito de guiar o processo de identificação de entidades candidatas a serem mantidas em cache.

Ao analisarmos o processo de arquivo de dados surgiram algumas questões que tiveram de ser respondidas:

- Qual a idade máxima dos dados arquivados? Devem ser mantidos todos os dados recolhidos desde a primeira publicação das aplicações? Ou apenas devem ser mantidos os dados mais recentes?

De modo a não sobrecarregar o sistema de armazenamento com informação desnecessária apenas deve ser guardada informação relevante para o processo de identificação de entidades candidatas a utilizarem cache. Em relação ao valor que deve ser estipulado para decidir o momento em que os dados recolhidos deixam de ser relevantes, este deve ser definido tendo em conta as características do ambiente analisado, como por exemplo o tempo de cada iteração no ciclo de desenvolvimento que tem normalmente uma duração entre 15 a 30 dias.

Posto isto, a nossa solução utiliza um limite configurável consoante o ambiente em que se encontra implementada.

4.3 Componente de Tomada de Decisões

Este componente é responsável por identificar as entidades de cada aplicação que devem ser automaticamente optimizadas e as que devem ser apresentadas como sugestões de optimização aos programadores. Para isso, esta componente obtém os dados arquivados sobre a utilização de cada entidade e aplica-lhes uma função heurística.

A função heurística tem como função analisar os dados recolhidos para uma determinada entidade da aplicação, recebidos como *input*, e produzir como resultado se deve ser introduzida cache sobre a entidade automaticamente, se deve ser sugerida ou se não se deve optimizar essa entidade (figura 4.4).

Para isso, a função heurística utiliza uma estratégia baseada em limites. Para cada dimensão de dados analisada devem ser definidos dois limites [1], representando os valores que ao serem atingidos indicam a decisão de optimização sobre a respectiva entidade:

Optimizar - a entidade deve ser seleccionada para introdução de cache automática.

[1] Note que considerando um domínio de valores contínuo apenas são necessários dois limites distintos para definir três intervalos

Sugerir - a entidade deve ser apresentada ao programador como sugestão de cache.

Ignorar - a entidade deve ser ignorada no processo de optimização.

Assim, o resultado final da função heurística é obtido pelo pior resultado do conjunto de resultados obtidos pelas várias dimensões de dados. Por exemplo, se três dimensões de dados analisados pela função heurística produzirem o resultado *Optimizar* e uma o resultado *Sugerir*, o resultado final é *Sugerir*.

A decisão sobre que dimensões de dados devem ser analisadas e que valores devem ser atribuidos como limites deve ser realizada de forma cuidada utilizando o máximo de informação possível, como por exemplo dados reais recolhidos para uma aplicação padrão onde se tenha um profundo conhecimento do modelo de dados e da utilização dessa aplicação. Este processo de afinação desses valores deverá seguir uma metodologia iterativa, de modo a obter-se os melhores limites para cada dimensão de dados, pois uma escolha pouco cuidada destes valores pode significar a introdução de falsas optimizações, ou seja, optimizações que possam piorar o desempenho das aplicações.

Para efeitos da nossa solução, onde pretendemos optimizar os acessos a bases de dados através da introdução de mecanismos de cache, as dimensões de dados que decidimos analisar na função heurística são, para cada entidade, a quantidade de acessos de leitura e escrita, a proporção entre esses valores e o tamanho dos dados guardados.

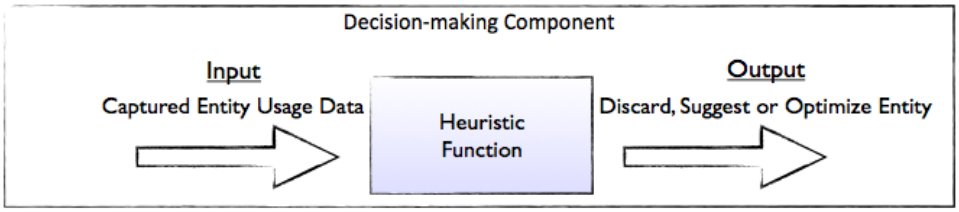

Figura 4.4: Diagrama de aplicação da função heurística sobre a informação recolhida para uma entidade de uma aplicação.

4.4 Mecanismo de Sugestões

O mecanismo de sugestões é um componente existente na ferramenta de desenvolvimento de aplicações e tem como função identificar, através da função heurística, as entidades que devem ser apresentadas aos programadores como candidatas à utilização de cache e apresentar essas sugestões de forma clara mas suficientemente informativa, para que os programadores possam ter confiança na sua decisão de aceitar ou rejeitar cada sugestão.

Este mecanismo deve também permitir ao programador analisar rapidamente os dados de maior relevância que levaram a função heurística a seleccionar tal entidade para ser sugerida.

Um aspecto importante deste ponto é o facto do programador se encontrar num ambiente diferente daquele onde se encontram os dados recolhidos para cada entidade (ambiente de desenvolvimento *VS* ambiente de produção). Assim, para se apresentar ao programador a infor-

mação recolhida sobre cada entidade é necessário introduzir um mecanismo de comunicação entre os dois ambientes que compreenda os seguintes passos principais (figura 4.5):

1. A ferramenta de desenvolvimento de aplicações envia um pedido ao ambiente de desenvolvimento pelos dados recolhidos sobre uma entidade de uma determinada aplicação.
2. O ambiente de desenvolvimento reencaminha o pedido para o ambiente de produção.
3. O ambiente de produção devolve ao ambiente de desenvolvimento os dados recolhidos para a entidade da aplicação pretendida.
4. O ambiente de desenvolvimento devolve os dados obtidos em ambiente de produção à ferramenta de desenvolvimento para que esta os possa apresentar ao programador.

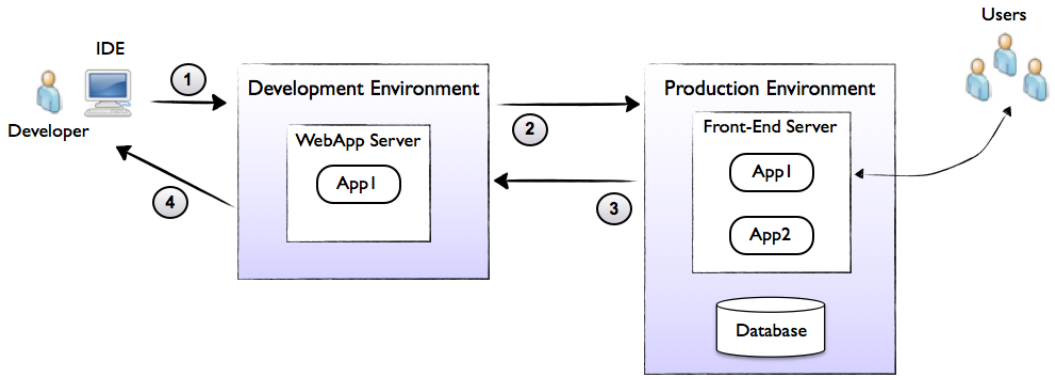

Figura 4.5: Processo de transporte de informação entre os ambientes de produção e de desenvolvimento.

Para além da função de apresentar as sugestões de cache e a informação de maior relevância recolhida sobre cada entidade, este componente é também responsável por obter a decisão dos programadores para cada sugestão apresentada e enviar essa informação para o compilador no momento de compilação das aplicações.

Todo o processo de identificação da sugestão, análise da informação recolhida e indicação da decisão tomada deve ser um processo óbvio e natural para o programador de modo a evitar desviar o foco de desenvolvimento para detalhes de implementação, melhorando assim a eficiência deste processo.

Este mecanismo pretende não só introduzir pressão sobre os programadores para que estes melhorem as suas aplicações, mas também que estes o façam com confiança, que aumenta com a análise de dados de execução das aplicações recolhidos em ambientes de produção.

4.5 Optimizador Guiado por Dados de Utilização

Depois de se ter obtido informação sobre quais as entidades das aplicações que devem ser optimizadas, que englobam não só aquelas que foram seleccionadas para optimização automática como também as correspondentes a sugestões aceites pelo programador, é então necessário

introduzir as respectivas optimizações. Neste contexo, as optimizações introduzidas consistem em utilizar padrões de cache *read-through* no código gerado, colocando em memória os resultados das consultas à base de dados que envolvam apenas entidades seleccionadas para serem optimizadas. Este processo de optimização será realizado em tempo de compilação e tem como objectivos diminuir os tempos de comunicação com serviços externos e reduzir a carga existente nos servidores de bases de dados para que este possa ser mais eficiente na resposta aos pedidos efectivamente necessários.

Para inserir mecanismos de cache o compilador deve ser alterado para que no momento de geração de código verifique se todas as entidades envolvidas em cada consulta estão marcadas para optimização e, caso estejam, este deve introduzir no código gerado mecanismos para que essas consultas utilizem a cache, onde serão mantidos os resultados dessas consultas. Um ponto de grande importância que deve ser tido em conta no momento de geração do código tendo em vista a utilização de cache diz respeito aos mecanismos de invalidação dos elementos em cache.

4.6 Invalidação de Cache

Em qualquer tipo de optimização que se introduza a uma aplicação é extremamente importante que essas optimizações não coloquem a aplicação num estado em que se possam obter comportamentos incorrectos. Ao introduzirmos mecanismos de cache o problema que surge diz respeito à consistência entre os dados mantidos na base de dados e em cache.

Apesar dos dados guardados nas entidades submetidas a optimizações sofrerem alterações pouco frequentemente, estes não são imutáveis e portanto em algum momento serão actualizados. Para que um programador aceite utilizar as optimizações da nossa solução é necessário que esta lhe apresente um mecanismo que garanta que caso os dados mantidos em base de dados sofram alterações, as entradas em cache que dependem desses dados deverão ser forçadas a realizar novas consultas à base de dados de modo a se obter a informação mais recente.

Assim, todos os elementos mantidos em cache devem possuir um mecanismo de invalidação muito eficiente, garantindo que apesar de momentâneamente poder existir inconsistências entre os dados em cache e em base de dados, rapidamente será reposta a validade desses mesmos dados.

Para que este mecanismo de invalidação não tenha grande impacto no desempenho das aplicações, este não deve seguir uma estratégia de *pooling*, onde a aplicação deve realizar consultas periódicas à base de dados para verificar se ocorreram alterações aos dados. Ao invés, deve seguir uma estratégia de notificações por parte da base de dados, tendo a aplicação apenas de receber e tratar essas notificações.

Alguns sistemas de gestão de bases de dados (SGBD) possuem mecanismos que permitem às aplicações registarem-se para receber notificações sempre que os dados mantidos em determinada entidade sofrem alterações. Caso o SGBD utilizado disponibilize este tipo de mecanismos este pode ser utilizado para conseguirmos o objectivo pretendido. Nos casos em que o SGBD utilizado não disponibiliza tais mecanismos pode-se conseguir um comportamento

idêntico através da criação de *triggers*. O impacto que estes triggers terão a nível da carga da base de dados deverá ser muito baixo pois estes apenas serão criados para entidades com baixa probabilidade de mudança.

A utilização duma abordagem deste tipo, em que o programador praticamente não interfere no processo de optimização das aplicações, permite aos programadores concentrarem-se no desenvolvimento de novas funcionalidades sabendo que estas serão automaticamente optimizadas à medida que for necessário. Contudo, as optimizações inseridas às aplicações devem ter um risco associado muito baixo para o programador, entendendo-se como risco a introdução de optimizações que causem um comportamento incorrecto ou uma diminuição no desempenho das aplicações.

4.7 Discussão

Durante a fase de desenho da nossa solução foram discutidos vários aspectos importantes. Um tema de bastante interesse foi *Sugerir VS Optimizar automaticamente*.

O que será melhor? Será que devemos optimizar automaticamente todos os pontos que considerarmos que deverão trazer ganhos no desempenho sem deixar o programador decidir se o quer fazer? Devemos apenas sugerir optimizações, nunca optimizando automaticamente, para que fique totalmente a cargo do programador a decisão e responsabilidade de aceitar a sugestão? Ou será que o ponto óptimo engola um misto destas duas hipóteses?

Todas estas hipóteses têm as suas vantagens e desvantagens, donde vamos apresentar alguns desses pontos. Se seguirmos a primeira abordagem, em que se optimiza tudo automaticamente, temos a vantagem de colocar a responsabilidade de optimização toda do lado da plataforma, não sendo necessário incomodar o programador para que este tome decisões deste tipo. Em contra partida, ao colocar toda a responsabilidade de optimização do lado da plataforma acresce também a responsabilidade desta por possíveis decisões tomadas de forma incorrecta.

A segunda abordagem, em que as decisões de optimização ficam sempre do lado do programador, permite à plataforma afastar por completo a responsabilidade da tomada de decisão para o programador, pois apesar da plataforma sugerir optimizações e apresentar os dados que sustentam a sua sugestão cabe sempre ao programador decidir se aceita ou não as sugestões apresentadas. Esta abordagem consegue com um custo muito baixo dar uma sensação de segurança e controlo ao programador, pois não o coloca à margem de um problema tão sensível como a escalabilidade de uma aplicação desenvolvida pelo mesmo.

Na terceira abordagem, que foi a utilizada na nossa solução encontramos um misto dos prós e contras de cada uma das abordagens anteriores, donde tentámos tirar o máximo partido das vantagens e diminuir ao máximo as desvantagens através de um processo de afinamento dos limites da função heurística bastante cuidado. No contexto deste trabalho optámos por seguir esta abordagem mista para exercitar ambas as alternativas, sendo que é conveniente considerar este aspecto como ponto de costumização futura.

5

Implementação e Validação da Solução em *OutSystems*

Neste capítulo encontra-se descrita a implementação e validação das várias componentes da nossa solução sobre a plataforma *OutSystems*. Começamos por apresentar a implementação do sistema de recolha de informação sobre a utilização das aplicações em execução. De seguida apresentamos a função heurística implementada, juntamente com o processo de validação dos limites nesta definidos. Depois é apresentada a implementação do mecanismo de sugestões e das optimizações introduzidas às aplicações, concluindo-se com a validação da nossa solução sobre a plataforma *OutSystems*.

5.1 Sistema de Recolha de Dados

Para se obter informação real sobre a utilização das aplicações desenvolvidas foram implementados dois mecanismos distintos:

- Sistema de *profiling*;

- Mecanismo de monitorização da base de dados por amostragem.

Estes dois mecanismos são utilizados em conjunto, de forma independente, de modo a obter o máximo de informação possível sobre a execução das aplicações. Como a informação recolhida pelo mecanismo de monitorização da base de dados está dependente do sistema de gestão de base de dados (SGBD) utilizado, que por vezes limitam a quantidade de informação disponibilizada, decidiu-se dotar o sistema *profiling* com capacidades para recolher alguma informação extra que possa ser utilizada como alternativa nos casos em que os SGBD assim o exijam.

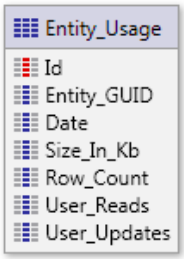

Figura 5.1: Diagrama da entidade *Entity_Usage*.

A implementação destes sistemas de recolha de informação foi realizada tendo em vista a recolha de informação necessária para guiar a identificação de entidades candidatas à utilização de cache e também a recolha de informação que permitisse ao programador avaliar o impacto da introdução dessas optimizações.

Como resultado desta recolha de informação pretendiamos obter um histórico sobre a utilização diária das entidades existentes no modelo de dados, guardando em cada entrada informação sobre a data em que os dados foram recolhidos, a chave da entidade analisada, o tamanho dos dados, o número de linhas nessa entidade, o número de acessos de leitura e o número de acessos de escrita sobre essa entidade. Para manter este histórico adicionámos a entidade *Entity_Usage* ao modelo de dados da plataforma (figura 5.1).

Vamos agora apresentar os dois mecanismos de recolha de dados implementados.

5.1.1 Sistema de *Profiling*

Tal como já foi referido em capítulos anteriores, esta componente é responsável por recolher dados reais sobre a execução das aplicações. O sistema de *profiling* usado na nossa implementação utiliza uma estratégia de recolha de dados baseada em instrumentação de código e corresponde a uma adaptação do sistema apresentado na secção 2.5, implementado durante a execução de um trabalho anterior [1], também realizado no contexto da *OutSystems*.

Inicialmente este sistema foi concebido para guardar informação sobre a execução de cada elemento da linguagem *OutSystems* presente na aplicação, o que deu origem à criação das estruturas de dados apresentadas na figura 5.2.

Para satisfazer as necessidades deste trabalho alterou-se a classe *ElementProfilerData* para guardar informação sobre quais as entidades afectadas por cada elemento, o tipo da operação realizada (leitura ou escrita), o número de vezes que a execução desse elemento foi satisfeita pela cache (cache *hits*) e o número de vezes em que o resultado não se encontrava em cache tendo sido necessário realizar-se um acesso à base de dados (cache *misses*). Como é de esperar, esta informação apenas é mantida para elementos que envolvam acessos a bases de dados, e, no caso da informação sobre o número de cache *hits* e cache *misses*, esta apenas é mantida para elementos guardados em cache. As estruturas de dados utilizadas pela nossa solução encontram-se apresentadas na figura 5.3.

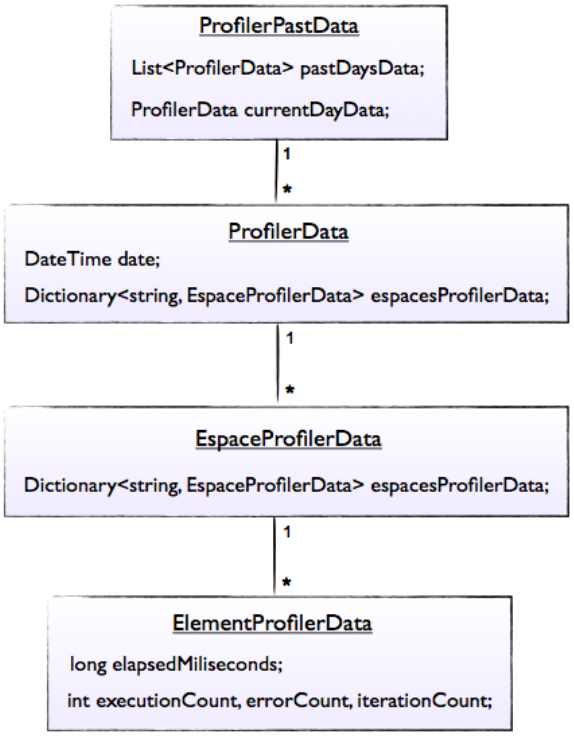

Figura 5.2: Estruturas de dados utilizadas pelo sistema de *profiling* inicial.

Um exemplo do código gerado para um elemento da linguagem após a instrumentação do seu código encontra-se apresentado na listagem 5.1.

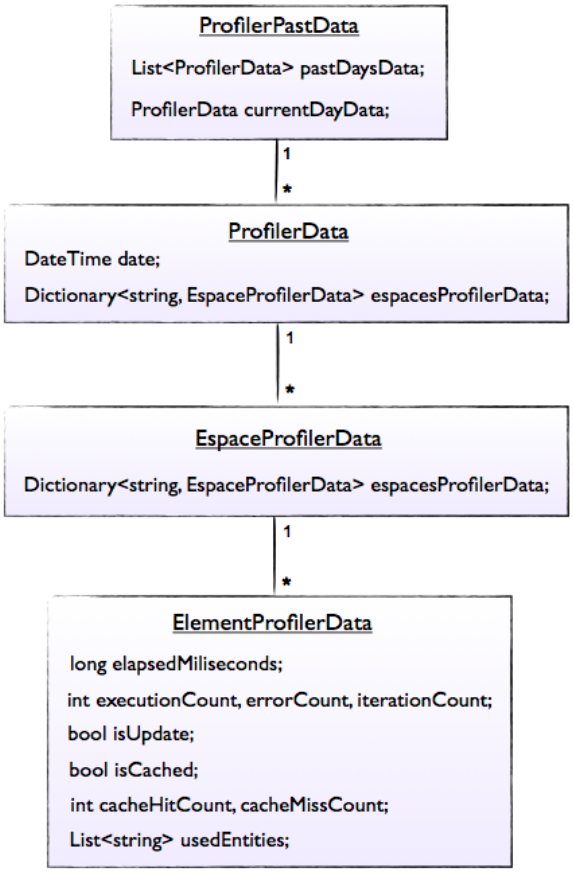

Figura 5.3: Estruturas de dados utilizadas pelo sistema de *profiling* implementado.

Listing 5.1: Código gerado para uma consulta com instrumentação de código pelo sistema de *profiling*.

```
public static void GetGroup(HeContext heContext, int inParamId, out RCGroupRecord result) {
   bool _profilerError = false;
   bool _profilerCacheHit = false;
   System.Diagnostics.Stopwatch _profilerStopWatch = new System.Diagnostics.Stopwatch();
   _profilerStopWatch.Start();
   try {
      RCGroupRecord temp = Cache.Get(cacheHash) as RCGroupRecord;
      if(temp == null) {

         // GET DATA FROM DATABASE

      } else {
         _profilerCacheHit = true;
         result = temp.First;
      }
   } catch(Exception) {
      _profilerError=true; throw;
   } finally {
      _profilerStopWatch.Stop();
      Profiler.ProfileElement("5177093c-02e4-4558-b630-dbed9288ce3e",
         "/Entities.UF3F+6+DdkqbfFaq98oRag/EntityActions.#GetEntity",
         _profilerStopWatch.ElapsedMilliseconds, _profilerError, 0,
         /*cachedAction*/true, _profilerCacheHit,
         new List<string> {"fbc55d50-83af-4a76-9b7c-56aaf7ca116a"},
         /*isUpdate*/ false);
   }
}
```

5.1.2 Serviço de Monitorização da Base de Dados por Amostragem

Este serviço foi concebido para produzir um histórico sobre a utilização diária das entidades existentes no modelo de dados através da análise de informação estatística mantida pelo sistema de gestão de bases de dados (SGBD) utilizado, o SQL Server. A informação que pretendíamos obter com este serviço era, para cada entidade, o número de linhas, o tamanho dos dados guardados e o número de acessos de leitura e de escrita. Para obter o número de linhas e o tamanho dos dados utilizou-se o procedimento *sp_spaceused* [26], disponibilizado pelo SQL Server. A figura 5.4 apresenta, a título de exemplo, a utilização deste procedimento para obter informação sobre a entidade *Entity*.

Para obter informação sobre o número de acessos de leitura e de escrita sobre cada entidade recorreu-se à análise da tabela de sistema *sys.dm_db_index_usage_stats* [27], onde é mantida informação estatística desde o arranque do sistema sobre a utilização de todos os índices existentes sobre tabelas da base de dados. A consulta realizada para obter a informação pretendida está apresentada na listagem 5.2.

```
EXEC sp_spaceused 'Entity'
```

Results | Messages

	name	rows	reserved	data	index_size	unused
1	Entity	1007	800 KB	368 KB	240 KB	192 KB

Figura 5.4: Exemplo de utilização do procedimento *sp_spaceused*.

Listing 5.2: Código SQL para obter a informação sobre a utilização de cada entidade.

```
1  select distinct Ent.PHYSICAL_TABLE_NAME as tablename,
2         Esp.ID as espaceId,
3        SUM (Usg.user_lookups) as user_lookups,
4        SUM (Usg.user_seeks) as user_seeks,
5        SUM (Usg.user_scans) as user_scans,
6        SUM (Usg.user_updates) as user_updates,
7        MAX (Usg.last_user_update) as last_user_update
8  from (dbo.ossys_Entity Ent inner join
9         dbo.ossys_Espace Esp on Ent.ESPACE_ID = Esp.ID) inner join
10        sys.dm_db_index_usage_stats Usg on
11          Ent.PHYSICAL_TABLE_NAME = OBJECT_NAME(Usg.object_id)
12 where Esp.IS_ACTIVE = 1 and Ent.IS_ACTIVE = 1
13 group by Ent.PHYSICAL_TABLE_NAME, Esp.ID
14 order by last_user_update desc
```

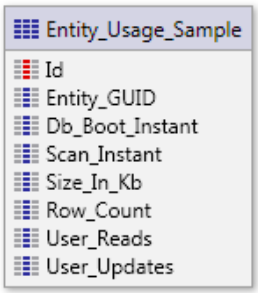

Figura 5.5: Diagrama da entidade *Entity_Usage_Sample*.

Devido a limitações do SGBD relativamente à persistência dos dados mantidos na tabela *sys.dm_db_index_usage_stats*, que são apagados sempre que o SGBD reinicia, foi necessária a criação da entidade auxiliar *Entity_Usage_Sample* (figura 5.5). O objectivo desta entidade é apenas tornar persistente a informação recolhida e mantida pelo SGDB na tabela *sys.dm_db_index_usage_stats*.

Vamos agora analisar a componente de tomada de decisão implementada na nossa solução.

5.2 Componente de Tomada de Decisões

Esta componente tem como função analisar a informação relativa à execução das aplicações e identificar quais as entidades que devem ser seleccionadas para serem automaticamente optimizadas e quais as que devem ser utilizadas no mecanismo de sugestões.

Para identificar as entidades a serem optimizadas são necessários dois passos. Primeiro é necessário obter-se a informação a analisar, sendo de seguida necessário aplicar uma função heurística a esses dados que indique o que fazer com cada entidade. Os dados utilizados pela função heurística no processo de análise são os dados guardados na entidade *Entity_Usage*, gerida pelo mecanismo de recolha de dados.

Tal como foi apresentado anteriormente, a entidade *Entity_Usage* mantém um histórico de informação diária sobre cada entidade. Nesta componente não se pretende analisar toda a informação recolhida, sendo apenas analisada a informação recolhida durante os últimos 30 dias. A razão pela qual se decidiu ignorar a informação mais antiga deve-se não só ao facto de ter sido considerado que uma amostra com esta dimensão teria informação suficiente para guiar o processo de identificação de entidades candidatas a utilizarem cache, como também devido ao padrão de utilização das aplicações poder alterar-se ao longo do tempo, o que poderia levar a uma produção de resultados incorrecta. Foi também imposto um limite inferior à quantidade de recolhas de dados feitas por entidade para garantir que a tomada de decisão é executada tendo uma quantidade suficiente de informação. Assim sendo, todas as entidades que não tenham registo de informação em pelo menos 15 dos últimos 30 dias são automaticamente descartadas deste processo. Os valores escolhidos têm em conta o facto de cada iteração do ciclo de desenvolvimento seguindo uma metodologia ágil ter normalmente uma duração entre 15 a 30 dias. Como este período de tempo pode variar de caso para caso, também os nossos valores podem ser configurados de modo a serem adaptados às circunstâncias.

O principal desafio encontrado na implementação desta componente foi o processo de decisão sobre quais os valores a atribuir aos limites da função heurística para cada tipo de dados recolhido. Como ponto de partida começámos por definir uma heurística inicial, baseada na intuição de um programador experiente, sobre a qual pudéssemos posteriormente trabalhar de forma iterativa, afinando os seus limites até se encontrar uma heurística que produzisse os resultados pretendidos. Apesar de terem sido estabelecidos valores para esses limites, todos esses limites podem ser configuráveis para satisfazer as necessidades doutros ambientes de produção. O processo de afinação desses limites e validação da função heurística é apresentado na secção seguinte.

5.2.1 Validação dos Resultados da Função Heurística

Como suporte ao processo de afinação dos limites utilizados pela função heurística foram utilizados dados reais recolhidos em ambiente de produção durante um período de 30 dias para a aplicação *IssuesManager*. Os dados recolhidos para algumas entidades desta aplicação encontram-se apresentados na figura 5.6.

Entity	total updates	total reads	reads / min(1, update)	size (KB)	rows
STATUS	0	2415636	2415636	8	38
PRIORITY	0	1873159	1873159	8	12
KIND	0	1049446	1049446	8	14
CASE_STATUS	0	673489	673489	8	6
CONTACT_TYPE	2	968386	484193	8	8
COMPONENT	1	478788	478788	8	150
CASE_SEVERITY	0	420019	420019	8	4
SEVERITY	0	320506	320506	8	4
PROBABILITY	0	320426	320426	8	5
FEATURE	0	230560	230560	24	72
HOLIDAYS	13	1157481	89037	8	75
TEST	9	115286	12810	688	4530
VERSION	315	2127105	6753	272	1614
REMINDER	0	940	940	8	25
CONTRACT	2513	972867	387	56	488
ISSUE_VERSION	26351	21718757	824	6968	211337
ATTACHMENT	19921	8651195	434	11752	179919
EMAIL	4664	268538	58	597968	149898
ISSUE	116177	5207014	45	299168	233568
HISTORY	145432	875696	6	432096	1768680
REPORT_LINE	372964	3341	0	2400792	30914511
TRACK	19299	174867	9	10680	389671
RELATED	70189	7330362	104	4216	153906
COMMIT	1693	131010	77	124464	155230
REPORT	1870	5044	3	176	6186
EMAIL_ATTACHMENT	7421	3244	0	17584	181808

Figura 5.6: Dados reais recolhidos para algumas entidades da aplicação *IssuesManager* durante um período de 30 dias.

Para nos ajudar no processo de afinação dos limites da função heurística decidimos recorrer a um programador experiente, responsável pelo desenvolvimento e manutenção do *IssuesManager*. Numa primeira fase pedimos-lhe que utilizando apenas o seu conhecimento profundo de modelação e uso da aplicação, identificasse quais as entidades que deveriam ser mantidas em cache. Sem qualquer acesso à informação recolhida pelo nosso sistema (figura 5.6), este identificou 10 dum total de 82 entidades como boas candidatas a utilizarem cache. De seguida apresentámos-lhe a informação recolhida sobre cada entidade da aplicação e pedimos-lhe que voltasse a repetir o processo anterior, tendo agora à sua disposição informação de produção sobre a utiização de cada entidade. Com base nesta informação, para além das 10 entidades atrás identificadas, este identificou outras 33 entidades como boas candidatas a utilizarem cache.

Tendo disponíveis os dados de utilização de cada entidade e o conjunto de entidades identificadas como boas candidatas a utilizarem cache pelo programador responsável pela aplicação estávamos então prontos para iniciar o processo de afinação dos limites da função heurística. Este processo passou por várias iterações até chegar ao ponto de obter uma forma conservadora de decisão para cada uma das dimensões recolhidas, apresentado na figura 5.7.

O resultado final obtido pela aplicação da função heurística foi uma surpresa para todos por descobrir um elevado número de entidades que tinham passado "despercebidas" numa análise

	ignore	suggest	automatic optimization
reads (daily)	< 20	20 - 100	> 100
updates (daily)	> 100	100 - 10	< 10
reads / min(1, updates)	< 100	100 - 500	> 500
size (KB)	> 8192	8192 - 64	< 64
rows	-	-	-

Figura 5.7: Limites utilizados pela função heurística implementada.

não quantitativa do uso das mesmas. Este facto foi bastante importante pois veio reforçar a nossa ideia da existência de entidades nos modelos de dados das aplicações, que apesar de não ser óbvio à primeira vista que estas devem ser mantidas em cache, após a análise dos dados reais de utilização das mesmas torna-se óbvio o uso deste tipo de optimizações. O resultado obtido mostrou que do total de 82 entidades existentes no modelo de dados da aplicação, 32 seriam seleccionadas para optimização automática e 11 seriam apresentadas ao programador como sugestão.

A figura 5.8 apresenta o resultado da aplicação da função heurística aos dados recolhidos para algumas entidades da aplicação *IssuesManager*. Em cada elemento das colunas utilizadas na função heurística encontra-se um circulo preenchido a cor, indicando o resultado da aplicação da função heurística àquele valor. O mapa de cores utilizado na imagem é o seguinte: vermelho - ignorar; amarelo - sugerir; verde - optimizar automaticamente.

	Entity	total updates	total reads	reads / min(1, updates)	size (KB)	rows
Automatic	STATUS	0	2415636	2415636	8	38
	PRIORITY	0	1873159	1873159	8	12
	KIND	0	1049446	1049446	8	14
	CASE_STATUS	0	673489	673489	8	6
	CONTACT_TYPE	2	968386	484193	8	8
	COMPONENT	1	478788	478788	8	150
	CASE_SEVERITY	0	420019	420019	8	4
	SEVERITY	0	320506	320506	8	4
	PROBABILITY	0	320426	320426	8	5
	FEATURE	0	230560	230560	24	72
	HOLIDAYS	13	1157481	89037	8	75
Suggest	TEST	9	115286	12810	688	4530
	VERSION	315	2127105	6753	272	1614
	REMINDER	0	940	940	8	25
	CONTRACT	2513	972867	387	56	488
Ignore	ISSUE_VERSION	26351	21718757	824	6968	211337
	ATTACHMENT	19921	8651195	434	11752	179919
	EMAIL	4664	268538	58	597968	149898
	ISSUE	116177	5207014	45	299168	233568
	HISTORY	145432	875696	6	432096	1768680
	REPORT_LINE	372964	3341	0	2400792	30914511
	TRACK	19299	174867	9	10680	389671
	RELATED	70189	7330362	104	4216	153906
	COMMIT	1693	131010	77	124464	155230
	REPORT	1870	5044	3	176	6186
	EMAIL_ATTACHMENT	7421	3244	0	17584	181808

Figura 5.8: Resultado da aplicação da função heurística a dados reais recolhidos para algumas entidades da aplicação *IssuesManager* durante um período de 30 dias.

5.3 Mecanismo de Sugestões

Um dos desafios deste trabalho consistia na criação de um mecanismo de sugestões que permitisse aos programadores não só visualizarem as sugestões recomendadas pela plataforma, mas também que disponibilizasse informação suficiente para que o programador pudesse aplicar essas sugestões com confiança.

O processo de identificação sobre que entidades devem ser sugeridas foi introduzido no processo de compilação das aplicações. Assim, durante a compilação é invocada a função heurística, sendo enviada uma mensagem para o *Service Studio* por cada entidade seleccionada para ser sugerida. Ao serem recebidas no *Service Studio*, estas mensagens são apresentadas no painel inferior, local onde são também apresentadas informações sobre o processo de compilação e publicação das aplicações. A figura 5.9 apresenta um exemplo das mensagens recebidas durante este processo, onde pode ser visualizada uma mensagem de sugestão de cache.

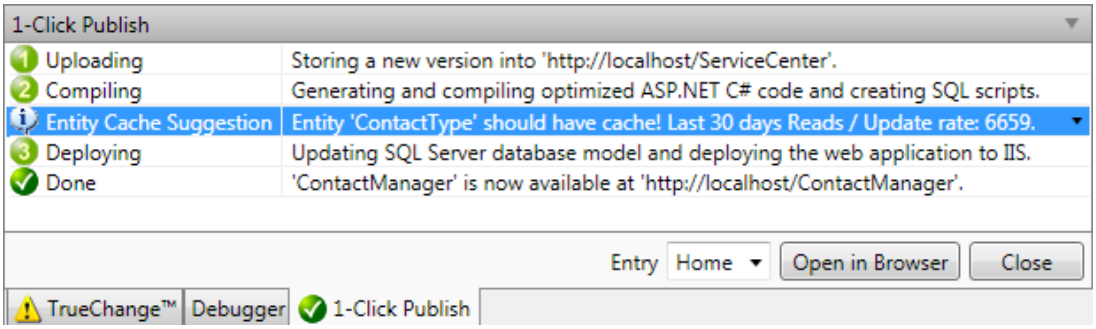

Figura 5.9: Mensagem de sugestão de cache apresentada no *Service Studio*.

Para que o programador possa aceitar as sugestões apresentadas com confiança, cada mensagem de sugestão disponibiliza um atalho no menu de opções que permite ao programador navegar para um ecrã onde são apresentados dados de utilização da entidade em questão. Nesse ecrã é apresentada a informação recolhida durante os últimos 30 dias para a entidade seleccionada, tal como o valor total de acessos de leitura e escrita durante esse período e os números de cache *hits* e cache *misses* das consultas sobre essa entidade. A figura 5.10 apresenta a informação disponibilizada aos programadores.

Como esta informação não se encontra disponível em ambiente de desenvolvimento, estando apenas disponível em ambiente de produção, foi necessário criar uma ligação entre estes dois ambientes de modo a que esta informação pudesse ser apresentada aos programadores. Para satisfazer este requisito foi criado um método, disponibilizado por um *Web Service* no *Service Center* (consola de administração da plataforma de suporte à execução das aplicações web *OutSystems*), responsável por obter essa informação e devolvê-la ao *Service Studio*. Este método pode ser dividido em duas partes principais: obtenção da informação recolhida pelo serviço de monitorização da base de dados e obtenção da quantidade de cache *hits* e cache *misses* das consultas sobre essa entidade.

O fluxo de acções desencadeadas por este método encontra-se representado na figura 5.11 e engloba os seguintes passos principais:

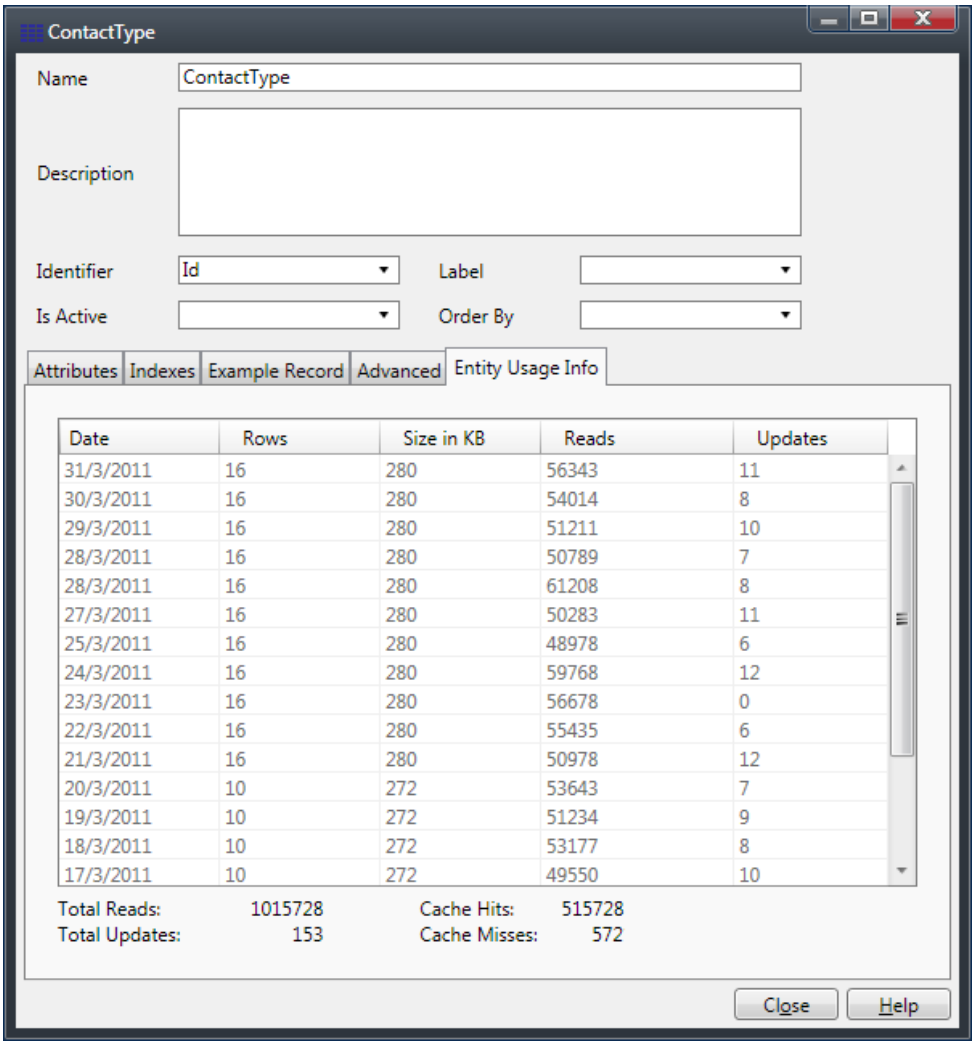

Figura 5.10: *Service Studio* com informação recolhida para uma entidade.

1. O *Service Studio* invoca um método sobre o *Service Center* em ambiente de desenvolvimento para que este lhe devolva a informação sobre a utilização de uma determinada entidade.

2. O *Service Center* em ambiente de desenvolvimento reencaminha o pedido para o *Service Center* em ambiente de produção.

3. O *Service Center* em ambiente de produção obtém da base de dados a informação mantida na entidade *Entity_Usage* para a entidade pretendida.

4. O *Service Center* em ambiente de produção comunica com todos os *Front-End Servers* através do serviço *Log Profiler*, para obter os dados relativos à quantidade de cache *hits* e cache *misses* recolhidos pelo Sistema de *Profiling* em cada um desses servidores.

5. Cada *Log Profiler* envia para o *Service Center* os dados recolhidos para a entidade pretendida, sendo o processo de agregação dos dados enviados pelos vários servidores efectuada pelo *Service Center*.

6. O *Service Center* em ambiente de produção agrega os dados devolvidos pelos servidores e os obtidos da base de dados e envia a informação resultante para o *Service Center* em ambiente de desenvolvimento.

7. Por último, o *Service Center* em ambiente de desenvolvimento recebe os dados de produção e devolve-os ao Service Studio que tratará de os apresentar ao programador.

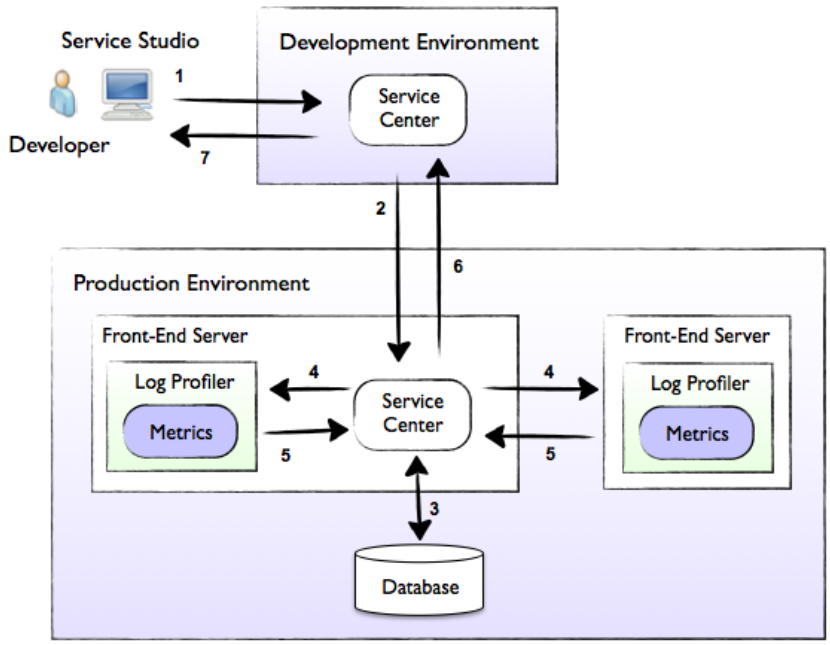

Figura 5.11: Processo de transporte de informação entre os ambientes de produção e de desenvolvimento em *OutSystems*.

Para além da informação global sobre a utilização de cada entidade foi também adicionada à informação já disponível sobre cada elemento da linguagem informação sobre a percentagem de cache *hits* desse elemento. A figura 5.12 apresenta a informação mostrada no *Service Studio* para uma *SimpleQuery*.

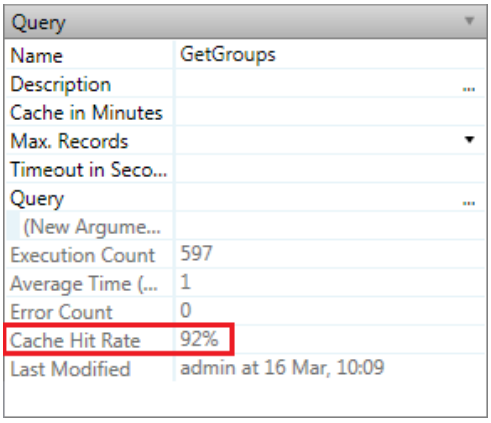

Figura 5.12: Cache hits apresentado no *Service Studio* para um elemento do tipo Query.

5.4 Optimizações

Apesar da abordagem utilizada no desenho da solução poder ser utilizada para introduzir vários tipos de optimizações, no contexto deste trabalho optámos por focar-nos apenas em optimizações através da introdução de mecanismos de cache. Este mecanismo de cache foi implementado ao nível do servidor de aplicações, seguindo uma estratégia de acessos de leitura do tipo *Read-Through*, tendo em vista a realização de cache em entidades do modelo de dados.

A escolha deste tipo de optimização foi suportada pelo facto de grande parte das aplicações empresariais centrarem os seus dados guardados em bases de dados, donde concluímos que reduzir os custos inerentes à obtenção desses dados seria um ponto que poderia trazer melhorias significativas no desempenho das aplicações.

5.4.1 Cache de Entidades

A introdução de cache sobre uma entidade é reflectida pela utilização de mecanismos de cache em todas as consultas à base de dados sobre essa entidade. Para consultas que envolvam várias entidades, estas apenas serão mantidas em cache caso todas as entidades envolvidas estejam marcadas como tal.

A activação desta optimização pode ser realizada de dois modos: manualmente pelo programador ou automaticamente (para as entidades seleccionadas pela função heurística).

Para que o programador tivesse a possibilidade de indicar que uma determinada entidade devê-se ser mantida em cache, foi adicionada uma nova propriedade à definição de entidade, tornando-a acessível para edição no painel de propriedades das entidades no *Service Studio* (Figura 5.13).

O processo de identificação das entidades a serem utilizadas na introdução automática de cache, tal como no processo de identificação de sugestões, é realizado durante o processo de compilação. Após a identificação das entidades alvo para optimização automática, o processo resume-se a activar, para cada uma dessas entidades, a propriedade que indica se esta deve ou não ser mantida em cache.

A activação desta propriedade traduz-se na utilização de cache dos resultados de todas as operações de leitura exclusivamente sobre entidades com esta propriedade activa, e também na utilização de mecanismos de invalidação de cache nas operações de actualização sobre essas entidades. A activação desta propriedade reflecte-se ao nível do código gerado para elementos do tipo *ComboBox*, *SimpleQuery*, *AdvancedQuery*, *GetEntity*, *DeleteEntity*, *CreateEntity*, *UpdateEntity* e *CreateOrUpdateEntity*.

5.4.2 Invalidação da Cache

Um grande desafio inerente à introdução dos mecanismos de cache era diminuir o risco de inconsistência entre os dados guardados em cache e os guardados na base de dados. Para diminuir este risco foi utilizado um mecanismo de invalidação de cache com dependências para a base de dados, recorrendo aos mecanismos de notificações do SQL Server implementados

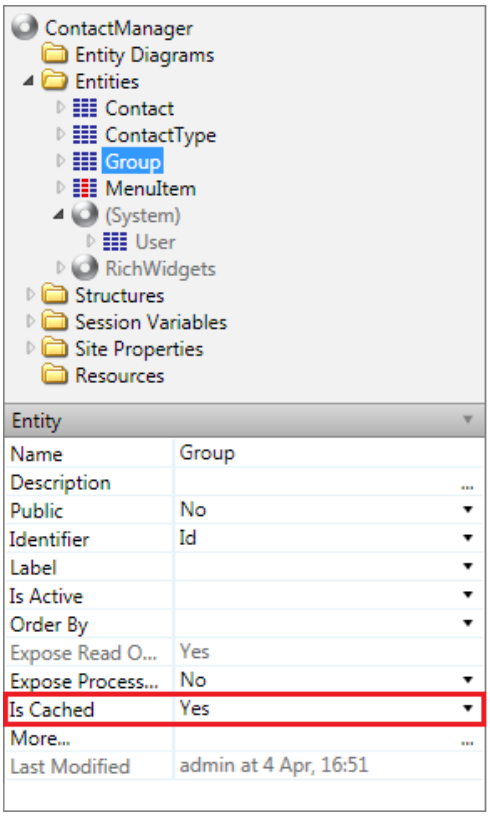

Figura 5.13: Propriedade de activação da cache para uma entidade no *Service Studio*.

pela classe *SqlDependency*. Assim, os dados guardados em cache apenas se encontram inconsistentes no intervalo de tempo entre a actualização dos dados na base de dados e o envio e processamento da respectiva notificação.

Como o desempenho deste mecanismo de notificações está dependente do número de dependências registadas no serviço, foi necessário encontrar uma estratégia que permitisse manter um número mínimo de dependências para a base de dados. Para isso, ao invés de se associar a cada elemento em cache uma dependência para a base de dados, foi inserido apenas um elemento em cache com este tipo de dependência por cada entidade com a propriedade de cache activa. Todos os outros elementos em cache são inseridos com dependências para os elementos que identificam as entidades envolvidas, ou seja, os elementos com dependências para a base de dados. Assim, apesar de apenas alguns elementos serem invalidados pelo sistema de notificações, todos os elementos que dependam destes também o são.

Apesar desta abordagem diminuir de forma muito significativa o número de dependências existentes entre os elementos em cache e as bases de dados, decidimos ir um pouco mais longe e tentar reduzir este número ainda mais. Para isso, em vez de existir em cache um elemento com dependências para a base de dados por cada entidade com a propriedade de cache activa, estes só existem enquanto forem necessários. Para isso, a inserção destes elementos em cache e criação das respectivas dependências foi implementado seguindo uma estratégia de criação a pedido.

Para que as aplicações reflectissem o comportamento atrás mencionado foi necessário alte-

rar também o código gerado para as entidades do modelo de dados.

5.4.3 Cache sobre Elementos da Linguagem

Na secção 2.6 apresentámos o mecanismo de cache existente na plataforma no momento em que iniciámos o nosso trabalho. Os mecanismos de invalidação deste sistema baseavam-se apenas em limites temporais ou através de acções de invalidação explícitas.

Durante o nosso trabalho decidimos alterar este mecanismo de cache para que, para além das políticas de invalidação existentes, existissem também dependências para os dados guardados na base de dados. O método adoptado para a criação das dependências foi o mesmo utilizado pelo mecanismo de cache implementado neste trabalho, sendo os elementos com esta propriedade activa inseridos com dependências para os elementos com as dependências para a base de dados.

5.5 Validação da Solução em *OutSystems*

Nesta secção vamos analisar os resultados obtidos com a implementação do nosso sistema de modo a validar a nossa solução. Começamos por apresentar os resultados obtidos através da análise do impacto da nossa solução sobre o código gerado para a aplicação *IssuesManager* (estudada na secção 1.2.2), seguindo-se os resultados obtidos pela realização de testes de desempenho sobre a mesma aplicação antes e depois da implementação do nosso sistema. Estes testes foram realizados num ambiente de produção simulado com o objectivo de validar e analisar o impacto que o nosso sistema teria num ambiente real.

5.5.1 Impacto no Código Gerado

Tendo sido validados os resultados obtidos pela função heurística implementada, era então necessário validar o impacto real que estes resultados teriam no desempenho das aplicações em produção. Começámos por realizar uma análise sobre o código gerado para a aplicação *IssuesManager* para identificar a quantidade de consultas à base de dados abrangidas pelo mecanismo de cache após terem sido aceites todas as sugestões feitas pela plataforma. De entre todas as consultas à base de dados decidimos dar algum destaque ao caso específico dos elementos do tipo *drop-down* por estes elementos utilizados com grande frequência nas aplicações web. Para realizarmos esta análise alterámos o compilador da plataforma de modo a contabilizar todos os elementos pretendidos.

Ao realizarmos esta análise estática sobre toda a aplicação, os resultados obtidos evidenciaram que dum total de 1543 elementos de consulta à base de dados, 493 utilizariam mecanismos de cache. Relativamente aos elementos do tipo *drop-down*, do total de 188 elementos, 125 seriam mantidos em cache.

Para estimarmos melhor o impacto que o nosso sistema teria sobre o desempenho da aplicação, realizámos uma análise focada apenas nas páginas da aplicação mais visitadas pelos utilizadores. Nesta análise foram observadas quatro páginas, responsáveis por cerca de 85 mil

pedidos dum total de 100 mil pedidos de páginas da aplicação analisados, tendo-se obtido os seguintes resultados. Do total de 133 elementos de consulta à base de dados, 67 utilizariam mecanismos de cache. Do total de 43 elementos do tipo *drop-down*, 32 seriam mantidos em cache.

Apesar dos resultados obtidos numa análise deste tipo não nos permitirem calcular os ganhos de desempenho da aplicação, estes podem ser usados como indicador da influência que o nosso sistema terá sobre a aplicação. Com base nestes resultados, nomeadamente dos resultados obtidos sobre as páginas mais visitadas pelos utilizadores da aplicação, podemos estimar que grande parte dos elementos que realizariam consultas à base de dados serão satisfeitos pela cache, melhorando assim o desempenho da aplicação.

5.5.2 Impacto no Desempenho das Aplicações

Depois de se terem analisado os resultados produzidos pela análise do código gerado, que foram bastante motivadores, foi necessário validar o impacto real que estes resultados teriam no desempenho das aplicações em produção.

Devido a limitações de tempo e risco não nos foi possível obter resultados reais sobre os ganhos de desempenho da aplicação directamente do sistema em produção. Assim, para a realização de testes de desempenho utilizámos uma réplica do sistema em produção e gerámos carga sobre a aplicação através de uma ferramenta que simula o acesso de vários utilizadores em simultaneo (*WAPT* [40]). Com a utilização desta ferramenta o nosso objectivo foi analisar três indicadores de desempenho da aplicação: quantidade de *pedidos respondidos por segundo* e *tempo de resposta* do servidor de aplicações, e quantidade de *pedidos por segundo* sobre o servidor de bases de dados.

Para a realização dos testes de desempenho foi utilizado um sistema composto por duas máquinas: uma para o cliente e para o servidor de aplicações e outra para o servidor de bases de dados. A arquitectura deste sistema encontra-se apresentada na figura 5.14.

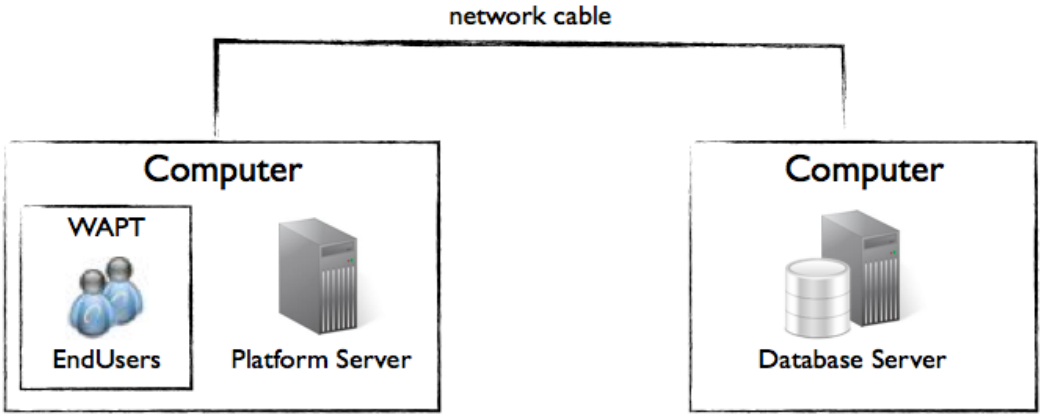

Figura 5.14: Arquitectura do sistema utilizado na realização de testes de desempenho em ambiente simulado.

Para que os resultados obtidos não fossem influenciados por factores externos decidimos utilizar duas máquinas dedicadas simplesmente a executar os testes, tendo sido efectuadas três repetições dos testes para garantirmos que os resultados obtidos não seriam fruto do acaso. As especificações físicas de cada máquina utilizada e suas configurações encontram-se apresentados na tabela 5.1.

	Client / Apps Server	**Database Server**
CPU	Quad Core 2.13 GHz	Dual Core 2.00 GHz
RAM	4.00 GB	4.00 GB
Operating System	Windows 7 Professional	Windows Server 2003
Web Server	Microsoft IIS 7.5	-
SGBD	-	SQL Server 2008

Tabela 5.1: Especificações *hardware* e configurações das máquinas utilizadas nos testes de desempenho.

De modo a tentarmos aproximar os resultados dos testes realizados aos que seriam obtidos num ambiente real, foram criados vários perfis de utilização da aplicação de acordo com os vários tipos de utilizadores reais do sistema, identificados através da análise dos registos de pedidos de páginas mantidos pela plataforma. Para podermos observar o desempenho do nosso sistema com diferentes tipos de carga os testes foram realizados aumentando progressivamente o número de utilizadores a aceder à aplicação até se atingir um limite de 100 utilizadores em simultâneo. Cada teste teve a duração de 10 minutos, sendo atingido o número máximo de utilizadores ao fim dos primeiros 5 minutos.

Realizando uma análise dos resultados obtidos (figura 5.15) sobre cada um dos indicadores analisados para o servidor de aplicações, podemos observar um aumento de cerca de 30% na quantidade de pedidos respondidos por segundo e uma diminuição de cerca de 20% no tempo de resposta médio. Analisando os resultados obtidos sobre o servidor de bases de dados observamos uma diminuição do número médio de pedidos de cerca de 60% (sem cache: 1386; com cache: 544). Este último resultado é sem dúvida bastante importante pois demonstra a grande quantidade de carga desnecessária efectuada sobre o servidor de base de dados. Num sistema em produção, em que as condições existentes na rede tendem em influenciar de forma mais significativa o desempenho das aplicações do que no ambiente usado nos nossos testes, acreditamos que os ganhos a nível do desempenho das aplicações possam ser ainda mais expressivos devido à grande redução da quantidade de acessos realizados à base de dados.

Figura 5.15: Valores médios dos resultados obtidos nos testes de desempenho da solução implementada em *OutSystems*.

6

Conclusões

As linguagens de domínio específico são bastante utilizadas em ambientes empresariais de modo a tornar mais eficiente o processo de desenvolvimento, colocando o foco dos programadores direccionado para os requisitos funcionais das aplicações. Contudo, apesar destas linguagens facilitarem o desenvolvimento de aplicações, estas tendem em resultar em aplicações pouco cuidadas a nível do consumo de recursos críticos, limitando a escalabilidade das mesmas. Se adicionarmos a este factor a adopção de metodologias ágeis, onde o desenvolvimento é focado na rápida produção de novas funcionalidades e posterior recolha de *feedback* dos utilizadores finais, os problemas relativos à falta de cuidado ao nível do consumo de recursos críticos têm uma grande importância, dado que apenas são detectados quando as aplicações em produção começam a falhar.

Neste trabalho apresentámos uma solução que permite às plataformas de suporte ao desenvolvimento e execução de aplicações a optimização e sugestão de optimizações guiadas por dados reais sobre a utilização das aplicações recolhidos em ambiente de produção. As optimizações efectuadas pela nossa solução visam diminuir a quantidade de acessos às bases de dados, e consequentemente a carga sobre esse servidor, através da introdução de mecanismos de cache para todas as consultas exclusivamente sobre entidades seleccionadas para optimização.

Na secção 1.3 apresentámos os vários desafios que se pretendiam ultrapassar com a nossa solução. De seguida são apresentadas as abordagens utilizadas para ultrapassar cada um dos desafios identificados.

Para que a recolha de informação durante a execução das aplicações não causasse um impacto significativo no desempenho das mesmas utilizámos duas abordagens distintas. A primeira foi utilizar técnicas de instrumentação para recolher alguma informação durante a interacção entre os utilizadores e aplicações. Ao recolhermos apenas um conjunto restricto de

métricas conseguidos ter um impacto mínimo no desempenho das aplicações, como foi provado em [1]. A segunda abordagem foi utilizar técnicas de amostragem sobre a base de dados, o que foi conseguido criando um serviço independente das aplicações em execução.

Um ponto importante da nossa solução consiste em apenas apresentar ao programador sugestões de cache que realmente melhorassem o desempenho das suas aplicações. Para conseguirmos obter esse resultado, no processo de afinação dos limites da função heurística foram utilizados dados reais de utilização de uma aplicação web empresarial e também a sabedoria de um programador experiente com conhecimento profundo sobre a aplicação analisada.

Para que os programadores tenham confiança e segurança na decisão de aceitar as sugestões apresentadas pela plataforma, estas são suficientemente informativas permitindo assim que os programadores se sintam confortáveis com as suas decisões.

Numa solução como a nossa, para que o programador tenha confiança nas sugestões de optimização dadas pela plataforma é necessário garantir que o risco associado às sugestões seja bastante baixo, o que no nosso caso traduz-se em garantir a consistência entre os dados mantidos em cache e na base de dados. Conseguimos isso através da implementação de mecanismos de invalidação criando dependências entre os dados em cache e os dados na base de dados, utilizando mecanismos de notificações, enviadas pela base de dados para as aplicações sempre que existam alterações aos dados com dependências para os elementos em cache, permitindo assim que as entradas inválidas sejam removidas da cache.

Apesar da plataforma utilizada para implementação do nosso sistema, a *Agile Platform* da *OutSystems*, ser um sistema maduro, com cerca de dez anos de existência e cerca de um milhão de linhas de código, conseguimos desenvolver a nossa solução sobre este sistema não comprometendo nem a dimensão nem a simplicidade do mesmo. Para a implementação deste trabalho foi necessário realizar alterações sobre várias componentes da plataforma, como por exemplo o compilador da linguagem, a plataforma de suporte à execução, o *Service Studio*, o *Service Center* e a base de dados da plataforma. Para atingirmos o resultado final apresentado neste documento o projecto passou por várias fases, desde a fase de estudo e discussão sobre possíveis optimizações a aplicar às aplicações e respectiva proposta de solução (cerca de 4 meses), passando pela fase de implementação e teste da solução proposta (cerca de 4 meses), terminando com a escrita do relatório final, que após um processo iterativo de revisão deu origem ao presente documento. Neste ponto foi essencial a ajuda da equipa de Investigação e Desenvolvimento da *OutSystems*, que através de troca de conhecimento permitiu uma rápida integração e aquisição de conhecimento sobre as várias componentes do sistema.

Realizando uma análise dos resultados obtidos num ambiente simulado em carga sobre o impacto da nossa solução no desempenho das aplicações conseguimos reduzir em cerca de 60% a quantidade média de pedidos sobre o servidor de bases de dados. Este valor foi bastante motivador visto que a carga exercida sobre o servidor de bases de dados foi identificado como sendo um ponto crítico em ambientes web empresariais. Outro indicador de desempenho analisado foi o tempo médio de resposta do servidor de aplicações. Ao contrário do indicador anterior, onde foram obtidos ganhos bastante significativos, neste apenas se observou uma melhoria de cerca de 20%, reduzindo-se o tempo médio de resposta de 7 para 5 segundos. Apesar

de se terem obtido melhorias no tempo médio de resposta, os valores obtidos ficaram um pouco aquém das nossas expectativas, o que nos permite afirmar que apesar da nossa solução ser interessante, ainda há espaço para se realizarem novos estudos sobre outro tipo de optimizações que, juntamente com a introdução de mecanismos de cache, possam melhorar de forma mais significativa o tempo médio de resposta do servidor de aplicações e por sua vez a usabilidade das aplicações.

Apesar da nossa solução estar focada para optimizações ao nível dos acessos a bases de dados, esta pode ser adaptada para inserir outro tipo de optimizações realizando pequenas modificações aos vários componentes desenvolvidos. Analisemos as modificações necessárias para introduzir por exemplo optimizações através de expansão de métodos *inlining*. Primeiro seria necessário alterar o Sistema de *Profiling* para recolher informação como o número de vezes que cada invocação de um método foi executada e o tempo médio gasto nessas invocações. Para isso seria apenas necessário adicionar os respectivos campos à estrutura *ElementProfilerData* usada pelo nosso sistema e seguir o padrão já utilizado na instrumentação de código para recolher a informação nos pontos pretendidos. De seguida seria necessário adicionar ao Componente de Tomada de Decisões uma função heurística adequada à optimização de *inlining*, para que fosse possível identificar os pontos que poderiam realmente melhorar o desempenho das aplicações. Para apresentar aos programadores as sugestões de *inlining* seria necessário criar um novo tipo de mensagens baseado no utilizado na nossa solução. Para apresentar a informação que suporta as sugestões poderiam ser adicionados dois campos ao Painel de Propriedades dos elementos da linguagem, do mesmo modo que apresentamos a percentagem de cache *hits* na nossa solução. Por último seria necessário necessário intervir no processo de geração de código. Neste ponto, para além de se introduzir a optimização de *inlining* seria necessário que este processo fosse realizado como primeiro passo do processo de optimizações, tornando assim visível às optimizações seguintes o corpo do método que seria invocado, melhorando assim o efeito dessas optimizações sobre as aplicações. Como podemos observar, apesar de ser necessário realizar alterações nos vários componentes da solução caso se pretenda estender o sistema para efectuar outros tipos de optimizações, estas alterações são simples e localizadas, tornando este processo relativamente simples.

Finalmente, realizando uma análise de todo o trabalho realizado, acreditamos que este projecto foi concluído com sucesso, tendo sido cumpridos os vários objectivos a que nos propusémos.

6.1 Trabalho Futuro

Através da análise dos resultados relativamente ao impacto da nossa solução no tempo médio de resposta do servidor de aplicações concluimos que ainda existe muito trabalho a ser feito de modo a melhorar o desempenho das aplicações.

Nesta secção apresentamos possíveis trabalhos futuros que poderiam trazer melhorias tanto a nível do desempenho das aplicações como a nível da experiência dos utilizadores finais das aplicações desenvolvidas com a plataforma.

Este trabalho pode também contribuir no desenvolvimento de futuros trabalhos, como por exemplo:

- Optimizar dinamicamente as aplicações de forma pró-activa introduzindo optimizações nos pontos seleccionados para optimização automática. Esta funcionalidade permite que as aplicações sejam optimizadas à medida que seja necessário sem necessidade de qualquer intervenção dos programadores.
- Sugerir ao programador segmentação de entidades, separando os atributos de baixa manutenção dos de alta manutenção com o objectivo de favorecer as consultas que apenas necessitam dos dados mantidos nos atributos de baixa manutenção através de mecanismos de cache. O processo de identificação de atributos de baixa e alta manutenção para cada entidade seria guiado pela análise de dados reais de utilização de cada entidade.
- Sugerir ao programador alterações ao conjunto de índices de modo a melhorar tanto o desempenho das aplicações quando os resultados das consultas não podem ser satisfeitos pela cache, como também diminuir a carga na base de dados associada à manutenção de índices desnecessários, através da eliminação de índices que não influenciem o desempenho das aplicações finais. Este processo de gestão de índices seria guiado pela análise de dados de utilização reais obtidos das bases de dados.
- Sugerir desactivação de cache para entidades cujos resultados obtidos pela optimização não se tenham traduzido em melhorias no desempenho das aplicações. Este processo poderia ser guiado pela análise da informação sobre a quantidade de cache *hits* e cache *misses* recolhida pelo nosso sistema para cada entidade.
- Optimizar as aplicações através de expansão de métodos *inlining* nos pontos das aplicações executados com maior frequência. Esta optimização abre espaço para que outras optimizações possam tornar-se mais eficientes eliminando as barreiras entre a invocação de um método e o seu corpo. A identificação dos pontos em que esta optimização deveria ser aplicada seria guiada por dados reais de utilização das aplicações, como por exemplo sobre a quantidade de execuções de cada método e o seu tempo médio de execução.
- Sugerir alterações à interface das aplicações para os caminhos de navegação mais frequentes de modo a melhorar a experiência dos utilizadores finais. As sugestões apresentadas seriam ao nível da dimensão, alinhamento e distância dos elementos da interface das aplicações, permitindo aos programadores destacarem os elementos com maior importância para os utilizadores. A identificação dos caminhos de navegação mais frequentes seria suportada pela análise de dados reais recolhidos durante a interação dos utilizadores com as aplicações.

Bibliografia

[1] Hugo Menino Aguiar. Profiling of real-world web applications. Master's thesis, Faculdade de Ciências e Tecnologias - Universidade Nova de Lisboa, October 2010.

[2] Alfred V. Aho, RavSethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, January 1986.

[3] Randy Allen and Ken Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufmann, 2002.

[4] Matthew Arnold, Stephen Fink, David Grove, Michael Hind, and Peter Sweeney. A survey of adaptive optimization in virtual machines. In *Proceedings of the IEEE*, volume 93. IEEE, February 2005.

[5] Matthew Arnold, Stephen Fink, David Grove, Michael Hind, and Peter F. Sweeney. Adaptive optimization in the jalapeño jvm. *SIGPLAN Not.*, 35(10):47–65, October 2000.

[6] Matthew Arnold, Stephen Fink, Vivek Sarkar, and Peter F. Sweeney. A comparative study of static and profile-based heuristics for inlining. *SIGPLAN Not.*, 35(7):52–64, January 2000.

[7] Matthew Arnold and Barbara G. Ryder. A framework for reducing the cost of instrumented code. *SIGPLAN Not.*, 36:168–179, May 2001.

[8] Thomas Ball and James R. Larus. Optimally profiling and tracing programs. *ACM Trans. Program. Lang. Syst.*, 16(4):1319–1360, July 1994.

[9] Calder Brad, Feller Peter, and Alan Eustace. Value profiling and optimization. *Journal of Instruction Level Parallelism*, March 1999.

[10] Bruno Cabral, Paulo Marques, and Luís Silva. Rail: code instrumentation for .net. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 1282–1287, New York, NY, USA, 2005. ACM.

[11] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. In *Proceedings of the 1996 annual*

conference on USENIX Annual Technical Conference, pages 13–13, Berkeley, CA, USA, 1996. USENIX Association.

[12] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design (4th Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[13] Adam Dingle and Tomás Pártl. Web cache coherence. *Computer Networks and ISDN Systems*, 28(7-11):907 – 920, 1996. Proceedings of the Fifth International World Wide Web Conference 6-10 May 1996.

[14] Alan Eustace and Amitabh Srivastava. Atom: a flexible interface for building high performance program analysis tools. In *Proceedings of the USENIX 1995 Technical Conference Proceedings*, TCON'95, pages 25–25, Berkeley, CA, USA, 1995. USENIX Association.

[15] M. Fowler and R. Parsons. *Domain-Specific Languages*. The Addison-Wesley Signature Series. Addison Wesley Professional, 2010.

[16] Markus Geimer, Sameer Shende, Allen Malony, and Felix Wolf. A generic and configurable source-code instrumentation component. In Gabrielle Allen, Jaroslaw Nabrzyski, Edward Seidel, Geert van Albada, Jack Dongarra, and Peter Sloot, editors, *Computational Science – ICCS 2009*, volume 5545 of *Lecture Notes in Computer Science*, pages 696–705. Springer Berlin / Heidelberg, 2009.

[17] Neal Glew and Jens Palsberg. Type-safe method inlining. *Science of Computer Programming*, 52(1-3):281 – 306, 2004. Special Issue on Program Transformation.

[18] James Gwertzman and Margo Seltzer. World-wide web cache consistency. In *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, pages 12–12, Berkeley, CA, USA, 1996. USENIX Association.

[19] Urs Hölzle and David Ungar. Reconciling responsiveness with performance in pure object-oriented languages. *ACM Trans. Program. Lang. Syst.*, 18(4):355–400, July 1996.

[20] Arun Iyengar and Jim Challenger. Improving web server performance by caching dynamic data. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, pages 5–5, Berkeley, CA, USA, 1997. USENIX Association.

[21] Peyton Jones and Simon Marlow. Secrets of the glasgow haskell compiler inliner. *Journal of Functional Programming*, 12(4-5):393–434, 2002.

[22] Chandra Krintz. Coupling on-line and off-line profile information to improve program performance. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, CGO '03, pages 69–78, Washington, DC, USA, 2003. IEEE Computer Society.

[23] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Sun Microsystems, Inc., second edition, 1999.

[24] J.D. Meier, Carlos Farre, Prashant Bansode, Scott Barber, and Dennis Rea. *Performance Testing Guidance for Web Applications*. Microsoft Corporation, September 2007.

[25] Ian Molyneaux. *The Art of Application Performance Testing*. O'Reilly Media, 2009.

[26] Microsoft. MSDN. sp_spaceused (transact-sql), June 2011. http://msdn.microsoft.com/en-us/library/ms188776.aspx.

[27] Microsoft. MSDN. sys.dm_db_index_usage_stats (transact-sql), June 2011. http://msdn.microsoft.com/en-us/library/ms188755.aspx.

[28] Steven Muchnik. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, August 1997.

[29] Jakob Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, Indianapolis, 1999.

[30] OutSystems. Outsystems website, June 2011. http://www.outsystems.com/.

[31] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Elsevier Science, March 2007.

[32] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35:374–398, December 2003.

[33] Massimiliano Poletto and Vivek Sarkar. Linear scan register allocation. *ACM Trans. Program. Lang. Syst.*, 21(5):895–913, September 1999.

[34] Joseph Ruzzi. *Oracle Coherence Getting Started Guide, Release 3.5*. Oracle and/or its affiliates, 2008.

[35] Michael L. Scott. *Programing Language Pragmatics*. Morgan Kaufmann, second edition, 2006.

[36] Avi Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts (8th Edition)*. John Wiley and Sons, Inc., July 2008.

[37] Alan Jay Smith. Cache memories. *ACM Comput. Surv.*, 14:473–530, September 1982.

[38] Toshio Suganuma, Toshiaki Yasue, and Toshio Nakatani. An empirical study of method in-lining for a java just-in-time compiler. In *Proceedings of the 2nd Java. TM. Virtual Machine Research and Technology Symposium*, pages 91–104, Berkeley, CA, USA, 2002. USENIX Association.

[39] Hugo Alexandre da Silva Veiga. Sistema distribuido de cache usando outsystems agile platform. Master's thesis, Instituto Superior Técnico - Universidade Técnica de Lisboa, Julho 2010.

[40] Web Application Testing WAPT, June 2011. http://www.loadtestingtool.com/.

[41] John Whaley. A portable sampling-based profiler for java virtual machines. In *Proceedings of the ACM 2000 conference on Java Grande*, JAVA '00, pages 78–87, New York, NY, USA, 2000. ACM.

[42] John Whaley. Partial method compilation using dynamic profile information. *SIGPLAN Not.*, 36(11):166–179, October 2011.

[43] Xiaolan Zhang, Zheng Wang, Nicholas Gloy, J. Bradley Chen, and Michael D. Smith. System support for automatic profiling and optimization. *SIGOPS Oper. Syst. Rev.*, 31(5):15–26, October 1997.

Dados de Utilização da Aplicação *IssuesManager*

As tabelas seguintes apresentam os dados de utilização recolhidos durante 30 dias para a aplicação *IssuesManager* utilizados como suporte ao processo de afinação dos limites da função heurística. De seguida apresenta-se também o resultado da aplicação da função heurística aos dados recolhidos para cada entidade. Relembrando o código de cores apresentado nas figuras: verde - optimizar automaticamente, amarelo - sugerir, e vermelho - ignorar.

Entity	total updates	total reads	reads / max(updates, 1)	size (KB)	rows
STATUS	0	2415636	2415636	8	38
PRIORITY	0	1873159	1873159	8	12
KIND	0	1049446	1049446	8	14
CASE_STATUS	0	673489	673489	8	6
CONTACT_TYPE	2	968386	484193	8	8
COMPONENT	1	478788	478788	8	150
CASE_SEVERITY	0	420019	420019	8	4
SEVERITY	0	320506	320506	8	4
PROBABILITY	0	320426	320426	8	5
PRODUCT_BACKLOG	0	237221	237221	32	716
FEATURE_2	1	230656	230656	32	1046
FEATURE	0	230560	230560	24	72
PROJECT	6	1357053	226176	16	86
CASE_STATUS_FLOW	0	131862	131862	8	17
HOLIDAYS	13	1157481	89037	8	75
CASE_ORIGIN	0	75716	75716	8	5
CASE_RESPONSE_TIME	0	41135	41135	16	143
CONTRACT_ENTRYTYPE	2	67968	33984	8	3
ENVIRONMENT	3	75649	25216	8	2
REMINDER_COMPONENT	1	20251	20251	8	158
REMINDER_PROJECT	1	20251	20251	8	25
REMINDER_KIND	0	20250	20250	8	18
CUSTOMER	35	580223	16578	40	494
CASE_PRODUCT	0	15484	15484	8	3
TEST	9	115286	12810	688	4530
AMT_STATUS	6	56438	9406	8	8
REMINDER_STATUS	25	225064	9003	8	39
RISK_LOOKUP	0	8450	8450	8	20
RISK_PRIORITY	0	8082	8082	8	4
VERSION	315	2127105	6753	272	1614
PROJECT_USER	56	330391	5900	16	495
AMT_DEPENDENCY	12	58804	4900	32	1005
FORBIDDEN	1	4485	4485	16	280
DEPENDENCY	143	598567	4186	184	6282
COMMUNICATION_LOG	23	58834	2558	32	13
COMPANY	0	1176	1176	8	4
REMINDER	0	940	940	8	25
ISSUE_VERSION	26351	21718757	824	6968	211337
PRODUCT_BACKLOG	0	499	499	8	27
ATTACHMENT	19921	8651195	434	11752	179919
CONTRACT	2513	972867	387	56	488
ITERATION	1359	409391	301	200	1011
USER_EMAIL	7	1933	276	48	837
COMMUNICATION_EXCLUSION	0	253	253	0	0
AMT_PRIORITY	0	190	190	8	3
FEATURE_1	28	5113	183	368	1374
USER_DEFAULTS	1314	224280	171	1152	20998
VERSION_NOTIFICATION	2	318	159	8	21
USER_ISSUE_PRIORITY	1203	146531	122	16	309
ACTIVITY_SUBSCRIPTION	4	466	117	8	91
VERSION_NOTIFICATION_PRIORITY	2	232	116	8	80
RELATED	70189	7330362	104	4216	153906
STATS_RE	2	171	86	8	20
COMMIT	1693	131010	77	124464	155230
SURVEY	0	71	71	8	17

Tabela A.1: Dados de utilização recolhidos durante 30 dias para as entidades da aplicação *IssuesManager* (1/2).

Entity	total updates	total reads	reads / max(updates, 1)	size (KB)	rows
SURVEYEXCEPTIONS	0	65	65	8	3
EMAIL	4664	268538	58	597968	149898
ISSUE	116177	5207014	45	299168	233568
CASE	14581	488904	34	19944	78426
SLA_REPORTING	0	23	23	8	9
FEATURE_5	0	14	14	8	3
ITERATION_HISTORY	90	874	10	2440	16586
TRACK	19299	174867	9	10680	389671
HISTORY	145432	875696	6	432096	1768680
SURVEYCASESQUEUE	95	564	6	8	9
STATISTICS_DAILY	612	2237	4	2072	9323
FEATURE_4	0	3	3	0	0
REPORT	1870	5044	3	176	6186
VERSION_BUDGET	252711	580924	2	3520	32661
EMAIL_ACCOUNT	90975	191790	2	8	15
STATSCHARTTYPES	140	294	2	8	10
FEATURE_6	0	2	2	8	192
PENDING_NOTIFICATION	69326	91117	1	8	0
SURVEYTRACK	32	34	1	16	388
ATTACHMENT_CONTENT	244	244	1	0	0
WHOLE_EMAIL	16	16	1	0	0
FEATURE_3	4	4	1	8	153
VERSION_BUDGET_EXCEL	0	1	1	7344	162
EMAIL_IGNORED	372	220	1	104	1656
SILENTISSUEQUEUE	18706	10356	1	72	0
EMAIL_ATTACHMENT	7421	3244	0	17584	181808
REPORT_LINE	372964	3341	0	2400792	30914511

Tabela A.2: Dados de utilização recolhidos durante 30 dias para as entidades da aplicação *IssuesManager* (2/2).

Entity	total updates	total reads	reads / update	size (KB)	rows
STATUS	0	2415636	2415636	8	38
PRIORITY	0	1873159	1873159	8	12
KIND	0	1049446	1049446	8	14
CASE_STATUS	0	673489	673489	8	6
CONTACT_TYPE	2	968386	484193	8	8
COMPONENT	1	478788	478788	8	150
CASE_SEVERITY	0	420019	420019	8	4
SEVERITY	0	320506	320506	8	4
PROBABILITY	0	320426	320426	8	5
PRODUCT_BACKLOG	0	237221	237221	32	716
FEATURE_2	1	230656	230656	32	1046
FEATURE	0	230560	230560	24	72
PROJECT	6	1357053	226176	16	86
CASE_STATUS_FLOW	0	131862	131862	8	17
HOLIDAYS	13	1157481	89037	8	75
CASE_ORIGIN	0	75716	75716	8	5
CASE_RESPONSE_TIME	0	41135	41135	16	143
CONTRACT_ENTRYTYPE	2	67968	33984	8	3
ENVIRONMENT	3	75649	25216	8	2
REMINDER_COMPONENT	1	20251	20251	8	158
REMINDER_PROJECT	1	20251	20251	8	25
REMINDER_KIND	0	20250	20250	8	18
CUSTOMER	35	580223	16578	40	494
CASE_PRODUCT	0	15484	15484	8	3
TEST	9	115286	12810	688	4530
AMT_STATUS	6	56438	9406	8	8
REMINDER_STATUS	25	225064	9003	8	39
RISK_LOOKUP	0	8450	8450	8	20
RISK_PRIORITY	0	8082	8082	8	4
VERSION	315	2127105	6753	272	1614
PROJECT_USER	56	330391	5900	16	495
AMT_DEPENDENCY	12	58804	4900	32	1005
FORBIDDEN	1	4485	4485	16	280
DEPENDENCY	143	598567	4186	184	6282
COMMUNICATION_LOG	23	58834	2558	32	13
COMPANY	0	1176	1176	8	4
REMINDER	0	940	940	8	25
ISSUE_VERSION	26351	21718757	824	6968	211337
PRODUCT_BACKLOG	0	499	499	8	27
ATTACHMENT	19921	8651195	434	11752	179919
CONTRACT	2513	972867	387	56	488
ITERATION	1359	409391	301	200	1011

Figura A.1: Resultado da aplicação da função heurística aos dados recolhidos para as entidades da aplicação *IssuesManager* durante 30 dias (1/2).

Entity	total updates	total reads	reads / update	size (KB)	rows
USER_EMAIL	7	1933	276	48	837
COMMUNICATION_EXCLUSION	0	253	253	0	0
AMT_PRIORITY	0	190	190	8	3
FEATURE_1	28	5113	183	368	1374
USER_DEFAULTS	1314	224280	171	1152	20998
VERSION_NOTIFICATION	2	318	159	8	21
USER_ISSUE_PRIORITY	1203	146531	122	16	309
ACTIVITY_SUBSCRIPTION	4	466	117	8	91
VERSION_NOTIFICATION_PRIORITY	2	232	116	8	80
RELATED	70189	7330362	104	4216	153906
STATS_RE	2	171	86	8	20
COMMIT	1693	131010	77	124464	155230
SURVEY	0	71	71	8	17
SURVEYEXCEPTIONS	0	65	65	8	3
EMAIL	4664	268538	58	597968	149898
ISSUE	116177	5207014	45	299168	233568
CASE	14581	488904	34	19944	78426
SLA_REPORTING	0	23	23	8	9
FEATURE_5	0	14	14	8	3
ITERATION_HISTORY	90	874	10	2440	16586
TRACK	19299	174867	9	10680	389671
HISTORY	145432	875696	6	432096	1768680
SURVEYCASESQUEUE	95	564	6	8	9
STATISTICS_DAILY	612	2237	4	2072	9323
FEATURE_4	0	3	3	0	0
REPORT	1870	5044	3	176	6186
VERSION_BUDGET	252711	580924	2	3520	32661
EMAIL_ACCOUNT	90975	191790	2	8	15
STATSCHARTTYPES	140	294	2	8	10
FEATURE_6	0	2	2	8	192
PENDING_NOTIFICATION	69326	91117	1	8	0
SURVEYTRACK	32	34	1	16	388
ATTACHMENT_CONTENT	244	244	1	0	0
WHOLE_EMAIL	16	16	1	0	0
FEATURE_3	4	4	1	8	153
VERSION_BUDGET_EXCEL	0	1	1	7344	162
EMAIL_IGNORED	372	220	1	104	1656
SILENTISSUEQUEUE	18706	10356	1	72	0
EMAIL_ATTACHMENT	7421	3244	0	17584	181808
REPORT_LINE	372964	3341	0	2400792	30914511

Figura A.2: Resultado da aplicação da função heurística aos dados recolhidos para as entidades da aplicação *IssuesManager* durante 30 dias (2/2).

B

Impacto da Solução no Desempenho das Aplicações *OutSystems*

Os gráficos seguintes foram obtidos durante os testes de desempenho realizados à nossa solução num ambiente simulado. A aplicação utilizada para os testes foi a *IssuesManager*, tendo sido utilizado a ferramenta WAPT para simular o ambiente de produção.

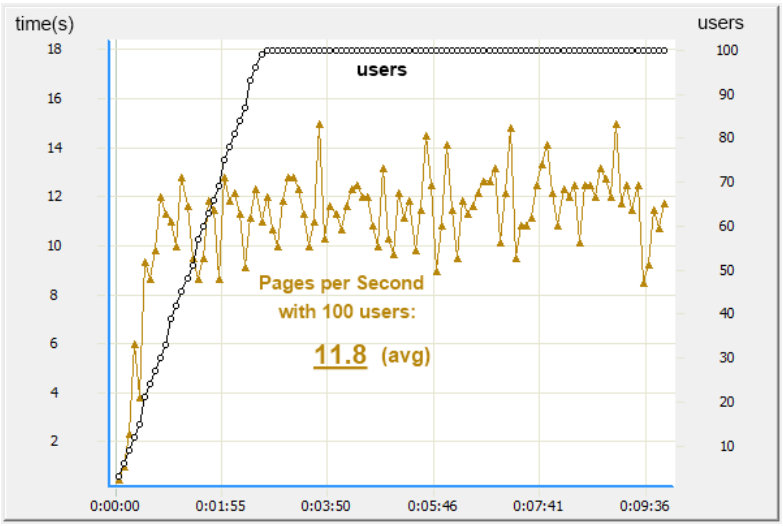

Figura B.1: Gráfico sobre a quantidade de *páginas respondidas por segundo*: teste com 100 utilizadores SEM cache.

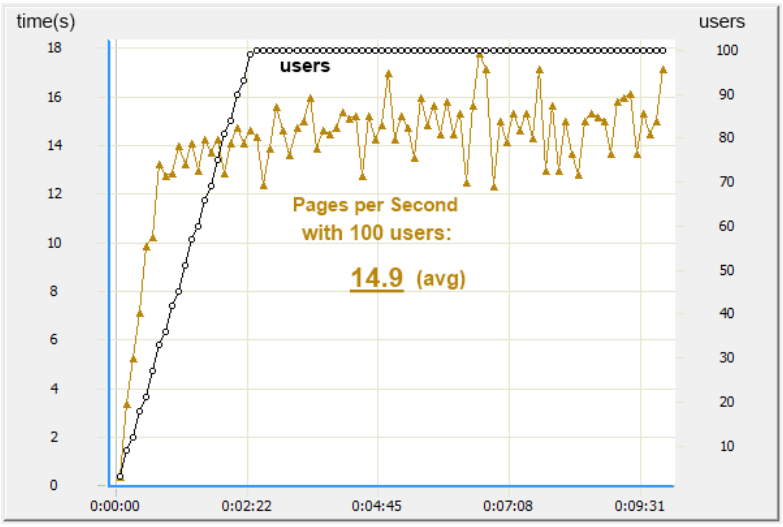

Figura B.2: Gráfico sobre a quantidade de *páginas respondidas por segundo*: teste com 100 utilizadores COM cache.

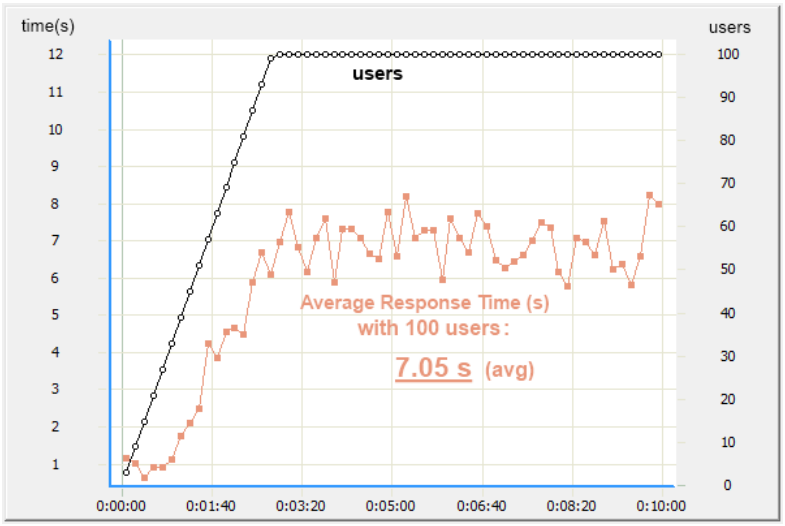

Figura B.3: Gráfico sobre *tempo de resposta médio*: teste com 100 utilizadores SEM cache.

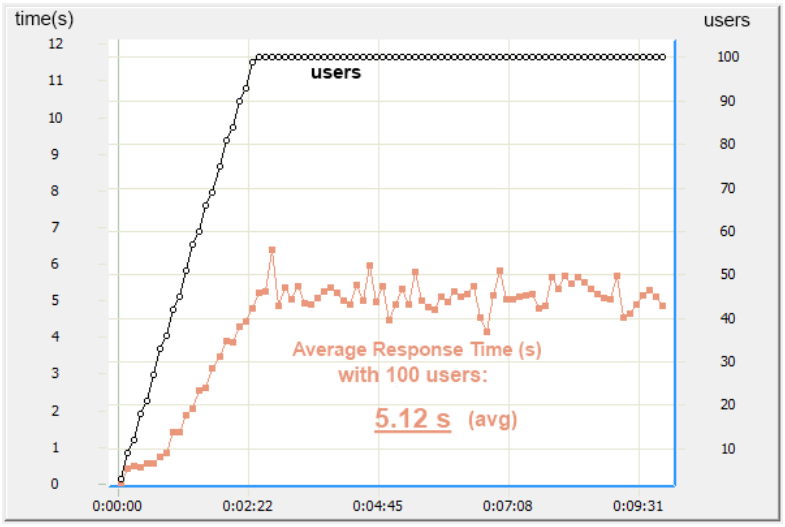

Figura B.4: Gráfico sobre *tempo de resposta médio*: teste com 100 utilizadores COM cache.

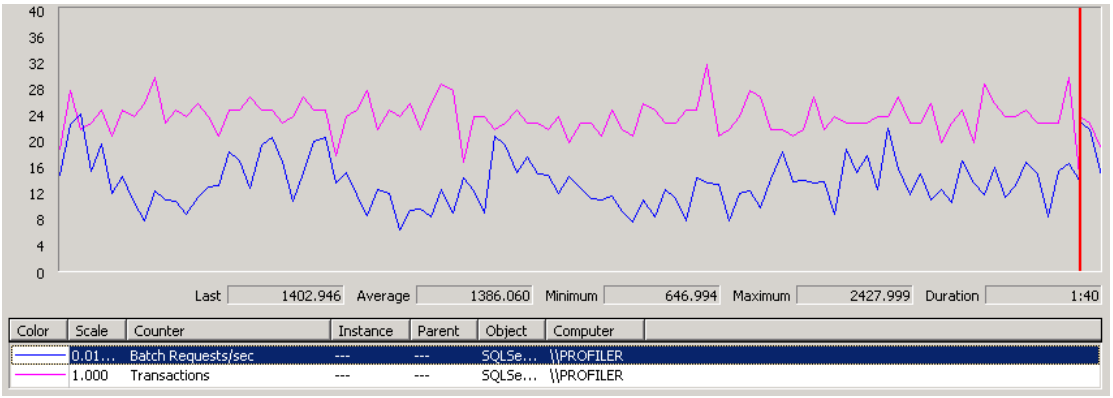

Figura B.5: Gráfico sobre a quantidade de *transações* e a *quantidade de pedidos por segundo* sobre a base de dados: teste com 100 utilizadores SEM cache.

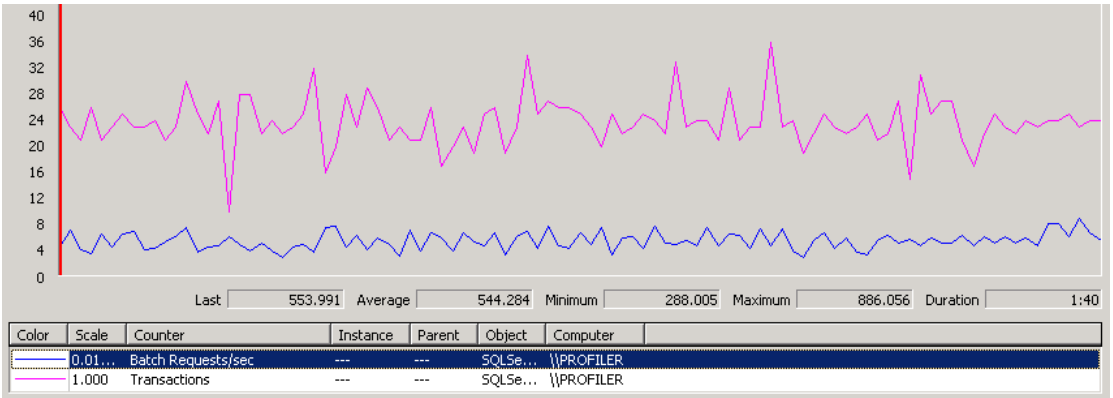

Figura B.6: Gráfico sobre a quantidade de *transações* e a *quantidade de pedidos por segundo* sobre a base de dados: teste com 100 utilizadores COM cache.