# Monitoramento de Metadados para Computação Ubíqua

**Caio Batista[1,2], Gustavo Alves[1], Everton Cavalcante[1], Frederico Lopes[3], Thais Batista[1]**

[1]DIMAp – Universidade Federal do Rio Grande do Norte – Natal-RN, Brasil
[2]Instituto Federal de Educação, Ciência e Tecnologia da Paraíba – João Pessoa-PB, Brasil
[3]ECT – Universidade Federal do Rio Grande do Norte – Natal-RN, Brasil

{caiosergiobatista, gustavoalvescc, evertonranielly,
fred.lopes, thaisbatista}@gmail.com

***Resumo.*** *Aplicações ubíquas são compostas por serviços fornecidos por diversos provedores de serviços, usam informações de contexto para realização das suas tarefas e operam em um ambiente extremamente dinâmico. Nesse cenário, é essencial monitorar os metadados relacionados a Qualidade de Serviço (QoS) e Qualidade de Contexto (QoC) para se garantir que a aplicação está usando serviços e informações de contexto com níveis de QoS e QoC que satisfaçam seus requisitos. Este artigo apresenta um módulo para monitoramento de metadados de QoS e QoC, usa ontologias para representação das informações, provê suporte a aferição de metadados e monitoramento síncrono e assíncrono. Este trabalho também ilustra o uso do monitor em uma aplicação ubíqua para a área de petróleo e gás e mostra uma avaliação de desempenho do mesmo.*

## 1. Introdução

A Computação Ubíqua (Weiser, 1991) utiliza uma grande variedade de dispositivos, sensores e redes integrados para formar um ambiente distribuído, altamente heterogêneo e integrado as atividades diárias dos usuários. Tipicamente, aplicações ubíquas recebem dados de sensores, de dispositivos e de provedores de serviços, gerenciam ações de usuários e oferecem suporte a mobilidade. Tais aplicações são compostas por *serviços,* fornecidos por diversos provedores de serviços, e são *cientes de contexto*, ou seja, usam *informações de contexto* para realização das suas tarefas. Uma informação de contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade, que pode ser uma pessoa, lugar ou objeto que é considerado relevante para a interação entre um usuário e uma aplicação (Dey *et al.*, 2001).

Nesse cenário onde as aplicações são compostas por dados de contexto e serviços provenientes de diversas fontes, é essencial se conhecer a qualidade das informações e serviços para que as aplicações possam usar aquelas que satisfaçam seus requisitos. Portanto, a seleção de um serviço específico, entre os serviços que oferecem a mesma funcionalidade porém disponibilizados por diferentes provedores, é realizada em função da qualidade das informações de contexto, denominada *Qualidade de Contexto* (QoC), e/ou da qualidade dos serviços prestados pela infraestrutura, denominada *Qualidade de Serviço* (QoS). Durante a execução das aplicações também é necessário garantir que os serviços e informações de contexto continuem atendendo aos requisitos de QoC e QoS da aplicação.

Uma forma que uma aplicação dispõe para detectar a qualidade dos serviços ou a indisponibilidade dos mesmos, ou ainda a qualidade de contexto, é consultando *metadados*. Metadados podem ser usados para descrever informações sobre as variáveis observáveis, seja sobre serviços ou sobre contexto, e são essenciais para o processamento e análise dos dados coletados. Por exemplo, em uma aplicação de *health care*, na qual é monitorada, entre outros parâmetros, a pressão arterial de um paciente, em dado momento, o valor deste parâmetro é "12/8", de modo que "12/8" é a informação de contexto. Entretanto, deve-se examinar outras características adicionais dessa informação através da descrição dos seus metadados em termos de QoC, tais como atualidade, grau de precisão, integridade, granularidade, etc. Da mesma forma, pode-se também examinar os metadados em termos de QoS, tais como tempo de resposta, latência, taxa de erro, escalabilidade, tempo médio entre falhas, etc. Esses metadados de QoS indicarão se os serviços utilizados correspondem aos padrões exigidos pela aplicação.

Aplicações ubíquas são inerentemente dinâmicas, uma vez que usam: (i) dispositivos móveis, que podem entrar e sair frequentemente da área de abrangência de uma dada rede; (ii) conexões sem fio, que estão sujeitas a interrupções e oscilações da intensidade do sinal transmitido, e; (iii) parâmetros físicos, como temperatura, pressão, localidade, que podem mudar frequentemente. Nesse contexto altamente dinâmico, as aplicações necessitam que, durante sua execução, os serviços e informações de contexto que elas usam mantenham níveis de QoS e QoC que satisfaçam seus requisitos. Portanto, é necessário prover um mecanismo para o constante monitoramento dos metadados, que deve prover suporte a aferição de metadados e ao monitoramento síncrono, i.e. realizado periodicamente em intervalos de tempo predefinidos, e/ou assíncrono, que em geral é disparado por um evento e não é regido por um intervalo de tempo bem definido.

Vários trabalhos (Truong *et al.*, 2006; Lin *et al.,* 2011; Zheng *et al.,* 2011) têm sido desenvolvidos para monitoramento de metadados, no entanto, nenhum deles reúne monitoramento de QoS e QoC e nem trabalham de forma síncrona e assíncrona. Nessa perspectiva, o objetivo deste artigo é propor um módulo de monitoramento que realiza aferição e monitoramento de metadados de QoS e QoC, opera de forma síncrona e assíncrona, bem como adota uma ontologia para representar os conceitos de forma não ambígua. Usando esse módulo, clientes (aplicações ubíquas) podem conhecer os metadados de QoC e QoS dos serviços e informações de contexto fornecidas por diversos provedores. Além de apresentar a arquitetura e funcionamento desse módulo, este artigo também apresenta o seu uso em um estudo de caso e uma avaliação de desempenho.

Este artigo está estruturado da seguinte forma. A Seção 2 apresenta o estudo de caso que será usado ao longo deste trabalho. A Seção 3 apresenta a arquitetura e funcionamento do monitor de metadados, bem como sua aplicação no estudo de caso. A Seção 4 apresenta a avaliação do mecanismo de monitoramento. A Seção 5 discorre acerca de trabalhos relacionados. Por fim, a Seção 6 contém conclusões e trabalhos futuros.

## 2. Estudo de caso: Monitoramento de poços de petróleo

O estudo de caso conduzido neste trabalho consiste em uma aplicação ubíqua da área de petróleo e gás natural que ilustra a necessidade do monitoramento da qualidade de serviço e de contexto dos serviços utilizados pela aplicação. Tal aplicação usa serviços de uma grande quantidade de provedores de serviços, e alguns desses serviços fornecem informações de contexto para as aplicações, e por esse motivo ela foi escolhida para

servir de estudo de caso no presente trabalho. A aplicação em questão monitora a carga de petróleo extraído de um poço a cada ciclo de movimento de uma unidade de bombeio (UB) mecânico para extrair petróleo. O propósito da aplicação é detectar a necessidade de trocar a configuração de operação da UB a fim de aumentar a produção de petróleo ou diminuir o desgaste do equipamento, ou ainda em casos que possam oferecer risco aos trabalhadores e ao meio ambiente. Desse modo, dependendo da carga de óleo extraído pela UB, a aplicação pode disparar ações para promover mudanças na configuração, notificar os responsáveis pelas decisões, ou ainda pará-la em casos mais graves.

Para operar corretamente, cada UB tem um *valor máximo* para a carga. Se esse valor máximo é alcançado abruptamente, a operação da UB deve ser paralisada para evitar danos à unidade. Além desse valor máximo, há um *valor intermediário* que representa que a unidade começa a operar com carga acima do ideal mas ainda não atingiu a carga máxima. Nesse caso, algumas ações são realizadas para prevenir que o valor máximo seja alcançado e a UB seja paralisada. Cada uma dessas ações pode ser realizada por mais de um serviço e, nesses casos, serviços com mesma funcionalidade normalmente são providos por diferentes provedores. O monitoramento de QoS e QoC associados aos serviços e informações de contexto provido por esses diferentes provedores é essencial para escolher qual o provedor mais adequado para a aplicação.

A Figura 1 apresenta os provedores de serviços usados na aplicação em questão. O provedor *WellDatabase* provê serviços que, de maneira assíncrona, fornecem o valor atual de vários parâmetros da produção de petróleo, dentre os quais a carga de petróleo em cada UB. *BMDimensioner* fornece serviços de gerenciamento de configurações do regime de operação da UB, sendo esse regime a relação entre o tamanho da haste da unidade de bombeio e os ciclos por minuto dessa haste. Assim, além de mostrar as possíveis configurações da UB, *BMDimensioner* também é responsável por atuar na operação da unidade, por exemplo, trocando seu regime ou paralisando a sua operação. *ChangeControlSystem* armazena e recupera mudanças realizadas nos equipamentos usados na exploração de petróleo. Especificamente nesse estudo de caso, esse sistema permite recuperar os regimes que foram adotados anteriormente em cada UB.

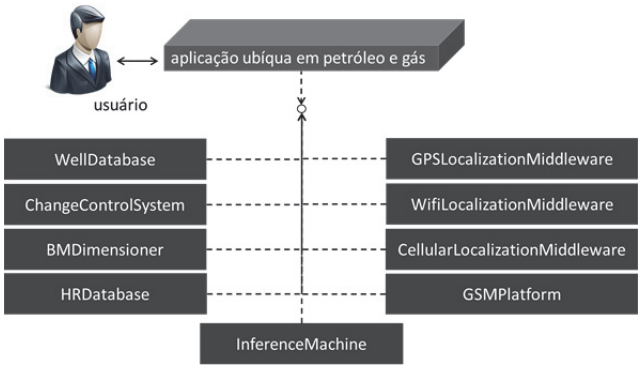


**Figura 1. Provedores de serviço do estudo de caso de monitoramento de poços.**

Os provedores *WifiLocalizationMiddleware, GPSLocalizationMiddleware* e *CellularLocalizationMiddleware* são plataformas responsáveis por fornecer serviços de localização dos técnicos espalhados pelos campos de petróleo. Esses serviços são usados para, caso seja necessária uma visita emergencial de técnicos ao poço monitorado, identificar quais técnicos estão mais próximos ao poço, visando um atendimento mais rápido. Embora tenham funcionalidades semelhantes, cada plataforma possui diferentes

níveis de qualidade que influenciam a seleção de serviços, de modo que um dos serviços equivalentes deve ser selecionado para uso pela aplicação. *HRDatabase* é um sistema que fornece informação sobre os funcionários, e.g. quais funcionários estão em serviço em um dado instante. Por fim, *GSMPlatform* é uma plataforma que fornece um serviço de envio de mensagens SMS.

A Figura 2 mostra um diagrama UML de atividades para a aplicação, no qual cada atividade deve ser realizada por pelo menos um serviço. A primeira atividade, *SubscribeBurden*, é responsável pela subscrição a um serviço de monitoramento da carga de petróleo na UB em questão. Se a carga atual encontra-se entre o valor intermediário e o valor máximo da carga para a UB, segue-se o fluxo de atividades *Flow1*, que engloba atividades que mudam automaticamente o regime de operação da unidade de bombeio; por outro lado, se o valor da carga excede o valor máximo, segue-se o fluxo de atividade *Flow2*. No fluxo *Flow1*, a atividade *SearchRegimeOptions* procura por possíveis opções de regimes de operação da UB e, em seguida, a atividade *SearchPreviousChanges* é realizada para descobrir os regimes usados previamente nessa UB. O próximo passo é trocar o regime e atualizar essa informação (atividades *ChangeRegime* e *UpdateChange*) no sistema de controle de mudanças, que armazena e recupera as trocas realizadas previamente. Por fim, é feita uma busca por técnicos disponíveis nas proximidades do poço de petróleo (atividade *SearchTechnicians*) e uma mensagem é enviada para eles para que possam verificar se tudo está funcionando conforme esperado (atividade *SendMsgToEmployee*). O fluxo *Flow2* descreve a situação na qual a carga é maior que o limite máximo da UB. Nesse caso, para evitar danos à unidade, a operação do poço é paralisada (atividade *StopWellOperation*) e depois é realizada uma busca pelo responsável pelo poço de petróleo (atividade *GetResponsibleEngineer*) e pelos técnicos que estejam nas proximidades (atividade *SearchTechnicians*). Por último, mensagens de advertência são enviadas a eles (atividade *SendMsgToEmployee*).

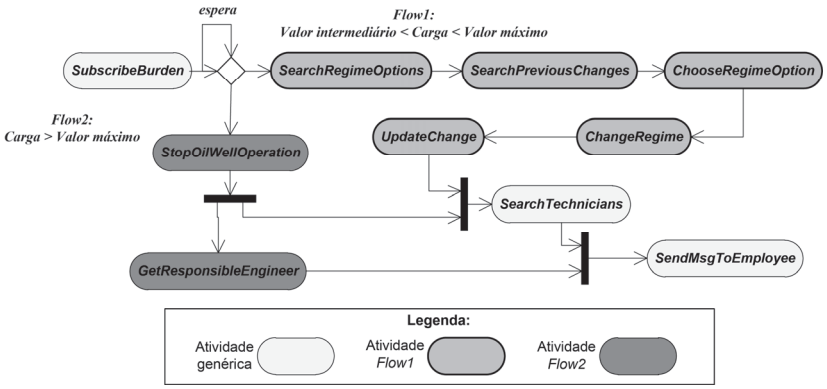


**Figura 2. Diagrama UML de atividades da aplicação.**

## 3. Monitor de metadados

Esta Seção apresenta a arquitetura e funcionamento do módulo de monitoramento (Seção 3.1) e como o mesmo é usado no contexto do nosso estudo de caso (Seção 3.2).

### 3.1. Arquitetura e Funcionamento

A Figura 3 ilustra a arquitetura do monitor que foi especificada visando a modularização dos seus componentes para que cada componente possa trabalhar de forma independente. O monitor oferece duas interfaces de comunicação, a saber, *ICliente*, para comunicação com clientes, e *IServidor*, para comunicação com plataformas provedoras de serviços.

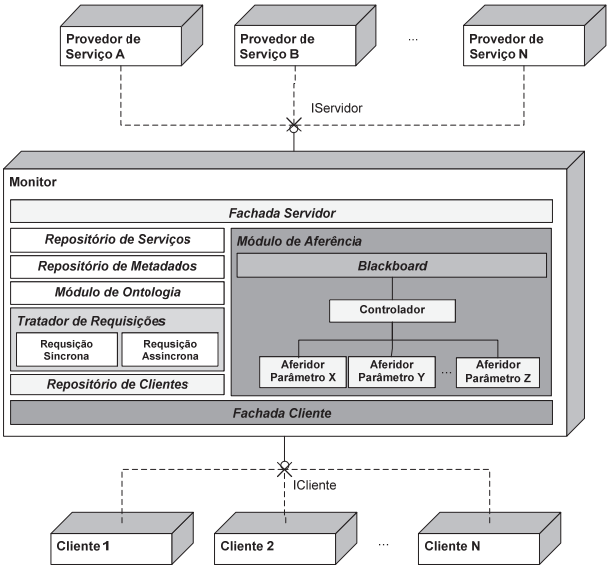


**Figura 3. Arquitetura do monitor.**

A *Fachada Servidor* modulariza a comunicação do monitor com os provedores de serviços, sendo responsável por registrar novos serviços no *Repositório de Serviços* e se comunicar os provedores. Quando uma dos provedores de serviços é registrado no monitor, a *Fachada Servidor* recebe os dados fornecidos por esse provedor e os repassa para o *Repositório de Serviços*. O *Repositório de Serviços* é responsável por armazenar as informações de todos os serviços que estão sendo monitorados pelo monitor e os parâmetros necessários para a comunicação com os mesmos. No monitor há duas formas de adicionar novos serviços ao repositório. Na primeira, o cliente faz uma requisição para recuperar dados que ainda não estão no repositório; nesse caso, a *Fachada Cliente* fornece os dados para armazenar o novo serviço no repositório. Na segunda forma, o próprio serviço registra-se no monitor, através da interface fornecida pela *Fachada Servidor*. A *Fachada Cliente* é responsável por fazer a comunicação do monitor com os clientes, que pode ser qualquer aplicação ubíqua ou *middleware* que necessita fazer uso do monitoramento de parâmetros de QoS/QoC. Através da interface *ICliente*, o cliente pode registrar-se no monitor e fazer requisições síncronas e assíncronas. Para registrar-se, o cliente passa suas informações para a *Fachada Cliente*, que repassa esses dados para o *Repositório de Clientes* para armazenamento, informações essas que são usadas quando o monitor precisa responder às requisições de tais clientes. Quando um cliente deseja fazer uma requisição ele passa os seus dados (nome, endereço IP e porta) e os dados do serviço monitorado (nome, endereço IP, porta, e lista de parâmetros de entrada). Caso a requisição seja assíncrona, é fornecida também uma determinada condição de retorno (nome do parâmetro, tipo de comparação, valor), de modo que o retorno desse tipo de requisição só é realizado quando essa condição de retorno é satisfeita. Por exemplo, utilizando o serviço *WifiLocalizationMiddleware* do estudo de caso do presente artigo, o formato de uma requisição síncrona poderia ser: <*Cliente1*, *187.61.111.165*, *8080*> (dados do cliente), <*WifiLocalizationMiddleware*, *186.236.203.76*, *8080*, *Field*, *OilWell*> (dados e lista de parâmetros do serviço). Caso essa requisição fosse assíncrona, todos esses dados seriam os mesmos acrescidos à condição de retorno, por exemplo, <*responseTime*, *greaterThan*, *100*>, significando que o cliente vai receber uma resposta quando o valor do parâmetro *responseTime* for maior que 100. De posse dos dados da requisição, a *Fachada Cliente* repassa a requisição ao *Tratador de Requisições* utilizan-

do referências para clientes e servidores (provedores) nos respectivos repositórios. Quando os dados da aferição são recuperados, a *Fachada Cliente* recebe os dados de no formato da ontologia junto com a referência para o cliente em questão.

O *Repositório de Metadados* é responsável por persistir todos os metadados de QoS e/ou QoC dos serviços que são aferidos pelo monitor e também metadados que são disponibilizados pelas plataformas de serviços. O *Módulo de Ontologia* é responsável por representar esses dados na forma de uma *ontologia*, conforme representado na Figura 4, na qual parâmetros de QoS e QoC estendem, respectivamente, as classes *QoS Parameter* e *QoC Parameter* definidas na ontologia. Uma ontologia é um modelo de dados que representa um conjunto de conceitos dentro de um domínio e os relacionamentos entre os mesmos, fornecendo expressividade formal e prevenindo ambiguidade nas interpretações semânticas da mesma informação. Por exemplo, Dobson *et al.* (2005) definem o parâmetro de QoS *ROCOF* (*rate of failure occurrence*), que possui a mesma definição do parâmetro *error rate* definido no trabalho de Guo *et al.* (2011) e que também é usado neste trabalho como a taxa de erros ocorridos em um determinado intervalo de tempo. Isso pode gerar um problema de interpretação que é solucionado com o uso de ontologias. Quando algum componente da arquitetura deseja receber os metadados no formato da ontologia, ele passa uma referência do serviço no *Repositório de Serviços* e o *Módulo da Ontologia* repassa ao *Repositório de Metadados,* que realiza uma busca e retorna os dados do serviço em questão. De posse desses dados, o *Módulo de Ontologia* realiza operações para representá-los no formato de ontologia empregada pelo monitor.

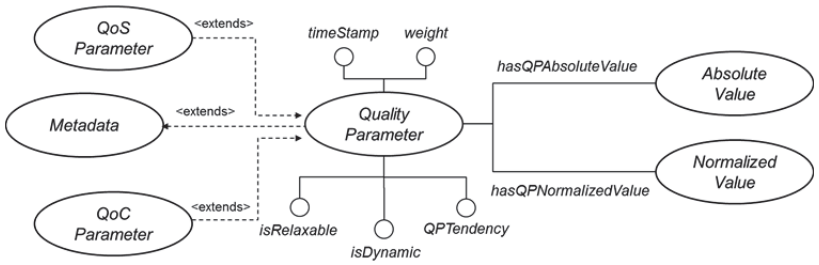


**Figura 4. Ontologia empregada pelo monitor para representação de dados.**

O principal componente do monitor é o *Módulo de Aferição,* responsável por aferir os metadados de QoS/QoC dos serviços armazenados no *Repositório de Serviços* e monitorá-los, sendo composto basicamente de três tipos de componentes: *aferidores*, *Blackboard e Controlador*. Cada aferidor é responsável por aferir um parâmetro de qualidade (QoS/QoC) específico a partir de informações capturadas através de requisições aos serviços monitorados pelo *Módulo de Aferição*. Essas informações são: (i) o tempo que a requisição levou para ser completada (*CompletedTime*); (ii) se o serviço estava disponível ou não (*Available*); (iii) o momento em que a requisição foi feita (*TimeStamp*), e; (iv) a data e hora de criação/sensoriamento das informações de contexto disponibilizadas pelo serviço, caso este seja um serviço que proveja informações de contexto, de modo que essa informação é importante por permitir inferir a idade das informações de contexto (*Age*) fornecidas pelo serviço. O componente *Blackboard* incorpora a ideia de um repositório de dados compartilhados (estilo arquitetural *Blackboard*), o que é interessante uma vez que os aferidores de diferentes parâmetros de QoS/QoC utilizam as mesmas informações listadas acima para calcular o valor desses parâmetros. Desse modo, o uso do componente *Blackboard* evita que um grande número de requisições aos serviços monitorados sejam realizados uma vez que, sem esse componente,

cada um dos aferidores precisaria requisitar os serviços isoladamente para ter acesso às referidas informações, o que poderia impactar negativamente no desempenho dos mesmos. Para evitar esse problema, o *Blackboard* mantem essas informações centralizadas de modo que cada aferidor fica apto a receber essas informações e calcular o parâmetro de qualidade a que se propõe aferir. Por exemplo, aferidores referentes a parâmetros de QoS como disponibilidade e taxa de erro podem fazer uso do histórico de dados armazenados no componente *Blackboard* sobre a disponibilidade do serviço para realizar a aferição. Por fim, a ideia do componente *Controlador* é controlar o acesso às informações armazenadas no *Blackboard* e as informações obtidas a partir da aferição dos parâmetros, de modo que os aferidores não conhecem a origem dos dados que eles utilizam para aferição, modularizando a arquitetura. O *Tratador de Requisições* é responsável por recuperar os dados de QoS/QoC através do *Módulo de Ontologia* e repassa-los para os clientes que estão solicitando os referidos dados. Quando um cliente faz uma requisição síncrona, a *Fachada Cliente* repassa a tarefa para o *Tratador de Requisições* que, por sua vez, recupera os dados atuais através do *Módulo de Ontologia* e responde à *Fachada Cliente*. Já quando uma requisição assíncrona é repassada ao *Tratador de Requisições*, este monitora se os dados satisfazem a condição de retorno (informada pelo cliente); nesse caso, o *Tratador de Requisições* monitora continuamente os dados a fim de identificar se a condição de retorno passou a ser satisfeita. Quando satisfeita, o *Tratador de Requisições* responde imediatamente à *Fachada Cliente*.

### 3.2. Monitorando provedores de serviço

Antes de iniciar o monitoramento dos provedores de serviço, é necessário definir dois intervalos de tempos no monitor. O primeiro intervalo, *TimeToRequest*, é o intervalo de tempo em que o *Módulo de Aferição* faz requisições aos provedores de serviço; já o segundo intervalo de tempo, *TotalTime*, é o intervalo de tempo em que as informações são consideradas recentes. Por exemplo, se o *TotalTime* for de dez minutos, então passado esse tempo, informações obtidas há mais de dez minutos começarão a ser ignoradas, visto que essas informações são consideradas desatualizadas e podem interferir nos cálculos de aferição dos parâmetros de qualidade. Uma vez definidos esses intervalos de tempo, o *Blackboard* faz requisições periódicas (de acordo com o *TimeToRequest*) aos serviços através da *Fachada Servidor*.

Após receber do *Repositório de Serviços* uma lista de referências aos serviços disponíveis (como os mostrados na Figura 1), o *Blackboard*, através da *Fachada Servidor*, faz requisições aos respectivos provedores dos serviços utilizando os dados do mesmo (endereço e lista de parâmetros), retornando o tempo que a requisição levou para ser completada (*CompletedTime*). Por exemplo, no caso do serviço *WellDatabase* que utiliza metadados de QoC, também é obtida a idade dessa informação (*Age*), caso a requisição tenha sido realizada com sucesso. Caso a requisição não tenha obtido sucesso, a *Fachada Servidor* lança uma exceção que é capturada pelo *Blackboard*. No estudo de caso, nenhuma plataforma fornece de antemão os metadados de QoS e QoC, de modo que, após cada requisição, o *Blackboard* armazena os dados da requisição (*CompletedTime*, *Available*, *TimeStamp*, *Age*). Caso a requisição tenha falhado *CompletedTime* possui valor -1, *Available* o valor falso, *Age* o valor nulo, o *TimeStamp* se mantém o mesmo. Se a própria plataforma de serviço já disponibilizar os metadados de QoS/QoC, o *Blackboard* repassa esses metadados para o *Módulo de Ontologia*, que faz a representação desses dados no formato da ontologia e então os envia para que o *Repositório de Metadados* faça o armazenamento dos mesmos.

Com os dados armazenados no *Blackboard*, o *Controlador* é invocado para acessar o histórico dos dados das requisições que estão no *Blackboard* e repassar esses dados para cada um dos aferidores. Depois que todos os aferidores terminaram sua aferição e retornaram o resultado ao *Controlador*, este repassa os dados para o *Repositório de Metadados* fazer o armazenamento dos mesmos. Essa execução é feita repetidamente com um intervalo de tempo definido por *TimeToRequest* e independe das requisições feitas pelos clientes, pois a ideia é que os dados de QoS/QoC já estejam armazenados antes dos clientes requererem. Dessa forma, o monitor conseguirá responder rapidamente às requisições dos clientes e poderá compartilhar dados caso dois clientes façam requisições ao mesmo serviço. Se por alguma razão os dados não estiverem disponíveis, e.g. quando for realizado o primeiro monitoramento, o *Tratador de Requisições* permanece em estado de espera até que os dados estejam disponíveis.

O estudo de caso ilustra a importância do uso de um monitor de metadados no momento de decidir qual serviço usar, p.ex., os provedores de serviço *WifiLocalizationMiddleware*, *GPSLocalizationMiddleware* e *CellularLocalizationMiddleware,* que atendem à atividade *SearchTechnicians* da Figura 2, são responsáveis por fornecer serviços de localização dos técnicos espalhados pelos campos de petróleo, cada uma usando tecnologias diferentes e com parâmetros de QoS/QoC diferentes. Sem os dados do monitoramento o cliente não saberá qual o serviço mais adequado a ser usado. Fazendo requisições síncronas ao monitor, o cliente pode descobrir quais os dados de QoS e QoC dos serviços que ele deseja usar, e sendo feitas requisições assíncronas ao monitor, o cliente pode decidir qual o melhor momento de usar uma determinada plataforma, p.ex., quando o tempo de resposta for menor que 50ms, ou a taxa de erro for menor que 10%.

## 4. Avaliação

Esta Seção apresenta uma avaliação do mecanismo de monitoramento proposto nesse trabalho usando uma perspectiva quantitativa, que tem como objetivo endereçar, através de experimentos computacionais, o tempo despendido para realizar as aferições de parâmetros de QoS e QoC. Para fins de avaliação foi utilizado o estudo de caso de monitoramento de poços de petróleo delineado na Seção 2. Os serviços utilizados no estudo de caso foram implementados como serviços Web utilizando a linguagem de programação Java e o *framework* Apache Axis[1], tendo sido implantados no servidor de aplicação Apache Tomcat[2] instalado em um computador com processador Intel® Core™ i7 2.7 GHz, 6 GB de memória RAM e sistema operacional Linux Ubuntu versão 11.10, que atuou como servidor ao qual as requisições eram realizadas. Nos experimentos, o mecanismo de monitoramento, sendo executado em um computador com processador Intel® Core™ i5 2.3 GHz, 4 GB de memória RAM e sistema operacional Mac OS X, realizava requisições aos serviços implantados no Apache Tomcat instalado no servidor remoto. Objetivando realizar tais experimentos em condições similares às observadas em um cenário real, o mecanismo de monitoramento e o servidor no qual os serviços estavam hospedados foram colocados em redes diferentes, de modo a não desprezar completamente a influência da rede no processo. Na avaliação quantitativa apresentada na Seção 4.2, para cada serviço foram realizadas dez execuções independentes para cada um dos seis parâmetros enumerados na Seção 4.1, a saber: *error rate*, *response time*, *MTBF*,

[1] Apache Axis – http://ws.apache.org/axis

[2] Apache Tomcat – http://tomcat.apache.org/

*MTTR*, *recoverable*, *uptime* e *freshness*. Nessas execuções, os valores escolhidos para o *TotalTime* e *TimeToRequest* foram 10 minutos e 5 segundos, respectivamente.

### 4.1. Parâmetros de QoS e QoC

No âmbito da Computação Ubíqua informações de contexto são coletadas de várias fontes, p.ex., podem ser fornecidas por usuários, obtidas de sensores, derivadas de múltiplas origens, etc. Os metadados referentes a parâmetros de QoC são associados a informações de contexto e têm o propósito de identificar a qualidade da informação, possuindo valores associados. Buchholz *et al.* (2003) enumeram alguns parâmetros de QoC, e.g. precisão, corretude, resolução, atualidade, etc. Para a avaliação realizada neste trabalho, consideramos o parâmetro de QoC *freshness* (atualidade), que expressa a idade da informação, ou seja, o tempo decorrido desde que a informação foi gerada. Quanto mais recente for uma informação certamente mais confiável ela será, visto que informações antigas podem estar desatualizadas. De maneira similar, os metadados referentes a parâmetros de QoS são associados aos serviços usados pelas aplicações ubíquas e têm como finalidade identificar a qualidade do serviço. Dentre os vários parâmetros de QoS enumerados em diversos trabalhos na literatura (e.g. Tran *et al.* (2009); Sathya *et al.* (2011)), para a presente avaliação consideramos seis parâmetros: (i) *response time* (tempo de resposta), que é o tempo decorrido desde o instante em que um cliente faz uma requisição até o instante em que este realiza o processamento da mensagem de resposta enviada pelo servidor; (ii) *MTBF* (*mean time between failures*), que é o tempo médio decorrido entre falhas no sistema durante sua operação; (iii) *MTTR* (*mean time to recovery*), que é o tempo médio decorrido entre uma falha no sistema e sua volta à operação (recuperação); (iv) *error rate*, que mede a taxa de erro na transmissão de dados ou no funcionamento de um serviço em um determinado período de tempo; (v) *recoverable*, que indica se houve recuperação de um serviço após a falha, e; (vi) *uptime*, que se refere ao tempo de funcionamento (i.e. disponibilidade) de um serviço.

Como mencionado anteriormente, para cada parâmetro existe um aferidor e consequentemente existe uma regra diferente de aferição: (i) o parâmetro *error rate* contabiliza o tempo total que o serviço ficou sem funcionar dentro do intervalo definido por *TotalTime*, de modo que esse tempo sem funcionar é dividido pelo *TotalTime* para definir o *error rate*; (ii) como o *error rate* é o valor percentual no qual o serviço apresentou falha dentro do intervalo de tempo *TotalTime*, o valor do parâmetro *uptime* é 1 menos o valor de *error rate*; (iii) para calcular o valor do parâmetro *MTBF*, o *TotalTime* é dividido pela quantidade de falhas contabilizadas pelo *Módulo de Aferição* nesse intervalo de tempo *TotalTime*; (iv) para calcular o valor do parâmetro *MTTR*, é contabilizado o tempo em que o serviço ficou sem funcionar dentro do intervalo de tempo *TotalTime* e depois esse valor é dividido por *TotalTime*; (v) o parâmetro *recoverable* verifica em todo o histórico se, depois de uma falha, o serviço voltou a funcionar; (vi) o parâmetro *response time* é uma média aritmética entre a quantidade de requisições feitas no período de tempo definido por *TotalTime* e o tempo que cada requisição levou para ser completada (*CompletedTime*), e, finalmente; (vii) para o parâmetro de QoC *freshness* é contabilizado o maior número de repetições do dado *Age* do serviço, que está no *Blackboard*, esse valor sendo então multiplicado por *TimeToRequest.*

### 4.2. Avaliação quantitativa

A Tabela 1 apresenta os tempos de aferição (em milissegundos) de cada um dos parâmetros de QoS considerados para cada um dos serviços do estudo de caso. O tempo de aferição de um dado parâmetro é basicamente o tempo despendido pelo respectivo aferidor para efetuar os cálculos dos valores desse parâmetro após os dados necessários para esses cálculos já estarem registrados no componente *Blackboard*. Como é possível observar claramente na Tabela 1, todos os tempos médios de aferição (reportados nas colunas *M*) não superam a ordem de 1 milissegundo, o que é algo extremamente benéfico no sentido de que, em termos de aferição dos parâmetros, o monitor não promove um impacto considerável. Por questões de espaço e dado que se tem apenas um serviço para o qual se pode obter o parâmetro de QoC *freshness*, a saber, *SubscribeBurden*, os valores referentes a tempo médio de aferição e desvio padrão não foram incluídos na Tabela 1. O tempo médio observado para a aferição desse parâmetro de QoC para tal serviço foi de 0,108 milissegundos, com desvio padrão de 0,015.

**Tabela 1. Tempo de aferição (em milissegundos) dos parâmetros de QoS considerados.**

| Parâmetros / Serviços | *error rate* | | *response time* | | *MTBF* | | *MTTR* | | *recoverable* | | *uptime* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **M** | **DP** | **M** | **DP** | **M** | **DP** | **M** | **DP** | **M** | **DP** | **M** | **DP** |
| *SubscribeBurden* | 0,968 | 0,285 | 0,047 | 0,010 | 0,148 | 0,052 | 0,130 | 0,042 | 0,049 | 0,020 | 0,052 | 0,024 |
| *SearchRegime* | 0,674 | 0,188 | 0,030 | 0,007 | 0,094 | 0,025 | 0,089 | 0,022 | 0,033 | 0,014 | 0,025 | 0,005 |
| *SearchChanges* | 0,810 | 0,224 | 0,041 | 0,008 | 0,135 | 0,063 | 0,119 | 0,035 | 0,053 | 0,011 | 0,036 | 0,008 |
| *ChooseRegime* | 0,928 | 0,244 | 0,053 | 0,010 | 0,137 | 0,037 | 0,128 | 0,036 | 0,057 | 0,010 | 0,035 | 0,005 |
| *ChangeRegime* | 0,690 | 0,194 | 0,035 | 0,008 | 0,104 | 0,030 | 0,100 | 0,029 | 0,037 | 0,019 | 0,033 | 0,009 |
| *UpdateChange* | 0,815 | 0,270 | 0,041 | 0,010 | 0,141 | 0,042 | 0,135 | 0,058 | 0,064 | 0,026 | 0,038 | 0,012 |
| *StopOilWell* | 0,785 | 0,298 | 0,034 | 0,013 | 0,106 | 0,043 | 0,102 | 0,040 | 0,037 | 0,021 | 0,027 | 0,006 |
| *GetRespEngineer* | 0,872 | 0,285 | 0,044 | 0,006 | 0,124 | 0,042 | 0,120 | 0,045 | 0,047 | 0,013 | 0,034 | 0,006 |
| *SearchTechGPS* | 0,806 | 0,243 | 0,040 | 0,005 | 0,116 | 0,035 | 0,113 | 0,035 | 0,051 | 0,009 | 0,035 | 0,005 |
| *SearchTechWifi* | 0,876 | 0,276 | 0,053 | 0,011 | 0,141 | 0,061 | 0,129 | 0,043 | 0,060 | 0,017 | 0,036 | 0,006 |
| *SearchTechCel* | 0,683 | 0,188 | 0,031 | 0,006 | 0,099 | 0,030 | 0,097 | 0,029 | 0,033 | 0,016 | 0,028 | 0,006 |
| *SendMessage* | 0,938 | 0,233 | 0,049 | 0,003 | 0,130 | 0,031 | 0,126 | 0,031 | 0,052 | 0,013 | 0,041 | 0,008 |

É importante ressaltar que os componentes que fazem as requisições e os componentes que fazem a aferição foram modularizados para que o tempo de processamento de um não influenciasse no do outro. Enquanto o componente *Blackboard* do *Módulo de Aferição* é responsável por fazer as requisições e capturar seus dados, o *Controlador* é responsável por capturar os dados do *Blackboard* para então invocar o aferidor de cada parâmetro. Assim, os tempos mostrados na Tabela 1 são os tempos despendidos por cada aferidor, independente do tempo necessário para fazer as requisições, fator que depende da rede, e do tempo para leitura e gravação de dados no *Blackboard*.

## 5. Trabalhos relacionados

Truong *et al.* (2006) apresentam uma ferramenta para monitoramento e análise de métricas de QoS de serviços de grades computacionais (*grids*) em tempo de execução. Os valores de QoS coletados pelos sensores para serviços individuais são enviados para um *middleware* que armazena os dados monitorados relativos aos serviços. Um raciocinador (*reasoning engine*) realiza análises de QoS com base em regras contidas em uma base de conhecimento, sendo possível estabelecer ações automáticas para reagir a mudanças nos parâmetros através de alertas ao cliente ou mesmo invocando interfaces de gerência de serviços para corrigir falhas. Apesar desse trabalho, tal como o nosso, se

propor a monitorar e analisar dados de QoS em tempo de execução, este não trata de QoC nem permite o atendimento a requisições síncronas e assíncronas. Da mesma forma, ambos os trabalhos permitem aos clientes recuperarem dados monitorados, uma vez que inclui um módulo de armazenamento. Como o nosso trabalho está centrado especificamente no monitoramento de QoS e QoC e na disponibilização desses dados para as aplicações, consideramos que ações associadas a reações automáticas a mudanças nos parâmetros de QoS e QoC são funções de outro módulo que foge ao escopo deste artigo.

Lin *et al.* (2011) propõem uma ferramenta genérica para a composição de serviços Web com base em parâmetros de QoS. Nessa ferramenta existe um componente chamado *QoS Manager*, responsável por gerenciar e monitorar metadados de QoS dos serviços publicados em um registro (*service registry*), de modo que esses parâmetros de QoS são utilizados para selecionar os serviços Web a serem incluídos nas composições de serviços a serem realizadas pela ferramenta, além de também fazer uso de ontologias. Entretanto, essa ferramenta não dispõe de módulos de gerência e monitoramento de metadados de QoC e de atendimento a requisições síncronas e assíncronas, funcionalidades oferecidas pelo nosso módulo de monitoramento.

Finalmente, Zheng *et al.* (2011) propõem uma ferramenta que realiza gerenciamento de QoC. Sensores fornecem informações de contexto para alimentar um repositório, através do qual os clientes acessam o módulo de gerência de contexto. Tal módulo tem as funções de manter um modelo de contexto, registrar e notificar outros módulos que recebem suas informações, prover acesso aos elementos de contexto e manter um registro dos componentes disponíveis. Raciocinadores de contexto filtram informações de contexto e notificam os componentes em relação às mudanças de contexto e o módulo de armazenamento guarda um histórico das informações de contexto obtidas. Além disso, a ferramenta provê gerenciamento de QoC usando ontologias. Entretanto, essa proposta é mais restrita no sentido de não dispor de monitoramento de QoS. De forma similar ao nosso monitor, essa ferramenta monitora continuamente as mudanças de contexto, ou seja, opera também de forma assíncrona.

## 6. Conclusão

Neste trabalho propomos um módulo de monitoramento de metadados de QoS e QoC que é um elemento essencial para aplicações ubíquas que necessitam escolher quais serviços e informações usarem com base nesses parâmetros em um ambiente extremamente dinâmico. Apresentamos a arquitetura e funcionamento desse módulo, que realiza aferição e monitoramento dos metadados, usa uma ontologia para representar os conceitos de forma não ambígua, e opera tanto de forma síncrona como assíncrona. Além disso, aplicamos o monitoramento em uma aplicação ubíqua na área de petróleo e gás natural que usa diferentes provedores de serviços e dados de contexto e realizamos uma avaliação de desempenho que mostrou que o tempo dispendido pelo monitor para realizar as aferições dos parâmetros de QoS e QoC não é significante.

Em termos de trabalhos futuros, pretendemos desenvolver um módulo de adaptação que interage com o módulo de monitoramento proposto para disparar adaptações dinâmicas nas aplicações quando os parâmetros de QoS e QoC não atenderem aos requisitos exigidos pela aplicação. Como a presente abordagem é genérica, permitindo que o monitor possa ser utilizado para aplicações em outros contextos, pretendemos também empregar esse monitor no contexto de uma aplicação em nuvem que utiliza

serviços de nuvens de diferentes provedores, permitindo que tais serviços sejam selecionados de acordo com os parâmetros de QoS/QoC monitorados.

Este trabalho está inserido no contexto dos Grandes Desafios da Computação, endereçando um problema relacionado com o quinto desafio "Desenvolvimento tecnológico de qualidade: Sistemas disponíveis, corretos, seguros, escaláveis, persistentes e ubíquos", que prevê desenvolvimento de modelos e ferramentas para o apoio a operação correta de aplicações ubíquas, incluindo mecanismos para garantir os níveis de qualidade dos serviços e dados de contexto exigidos pela aplicação. Esse tópico é especialmente importante dado que falhas nos serviços ou uso de dados de contexto fora dos limites estabelecidos pela aplicação podem ter consequências catastróficas ou prejuízos enormes. Por exemplo, em aplicações ubíquas da área de saúde podem-se ter perdas de vidas se não houver garantia de que os dados vitais do paciente (dados de contexto) estão sendo providos com uma taxa de atualização (parâmetro de QoC) dentro de limites rígidos.

## 7. Agradecimentos

À Agência Nacional do Petróleo e Biocombustíveis (ANP), através do PRH-22, pelo apoio financeiro.

## Referências

Buchholz, T.; et al. (2003) Quality of Context: What it is and why we need it. *Proceedings of the 10th Workshop of the HP OpenView University Association*.

Dey, A.; et al. (2001) A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Journal of Human-Computer Interaction* 16(2), pp.97–166.

Dobson, G.; Lock, R.; Sommerville, I. (2005) Developing an ontology for QoS. *Proceedings of the 5th Annual DIRC Research Conference*, pp.128–132.

Guo, G.; Yu, F.; Xie, D. (2011) A method for semantic Web service selection based on QoS ontology. *Journal of Computers* 6(2), pp.377–386.

Lin, C. et al. (2011) A relaxable service selection algorithm for QoS-based service composition. *Information and Software Technology* 53(12), pp.1370–1381.

Sathya, M. et al. (2011) Evaluation of QoS based Web service selection techniques for service composition. *International Journal of Software Engineering* 1(5), pp.73–90.

Tran, V.; et al. (2009) A new QoS ontology and its QoS-based ranking algorithm for Web services. *Simulation Modeling Practice and Theory* 17, pp.1378–1398.

Truong, H.L. et al. (2006) Towards a framework for monitoring and analyzing QoS metrics for grid services. *Proceedings of the 2nd IEEE Conference on e-Science and Grid Computing*, USA, IEEE Computer Society.

Weiser, M. (1991) The computer of the Twenty-First Century. *Scientific American* 265(3), pp.94–104.

Zheng, D.; Wang, J. (2011) Research of the QoC based middleware for the service selection in pervasive environment. *International Journal of Information Engineering and Electronic Business* 3(1), pp.30–37.