Um Sistema para Monitoramento de Metadados para Aplicações Ubíquas

Gustavo Alves, Thais Batista

DIMAp – Universidade Federal do Rio Grande do Norte – Natal-RN, Brasil

gustavoalvescc@gmail.com, thais@ufrnet.br

Resumo. *Aplicações ubíquas utilizam diversos provedores de serviço e informações de contexto para realização das suas tarefas e operam em um ambiente dinâmico e heterogêneo. 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. Nesse trabalho é apresentado o QoMonitor, um sistema de monitoramento que realiza aferição e monitoramento de metadados de QoS e QoC, opera de forma síncrona e assíncrona, e adota uma ontologia para representação dos conceitos de forma não ambígua. O QoMonitor foi integrado com o OpenCOPI, uma plataforma que facilita o desenvolvimento e execução de aplicações ubíquas. Este trabalho também ilustra o uso do monitor em uma aplicação ubíqua para monitoramento de poços de petróleo. Os resultados da avaliação de desempenho mostraram que o uso do QoMonitor pelo OpenCOPI não gerou um impacto considerável e possibilitou ao OpenCOPI trabalhar com valores reais de QoS e QoC.*

Abstract. *Ubiquitous applications use several service and context providers for performing their tasks and work in a dynamic and heterogeneous environment. In such scenario, it is essential to monitor metadata related to Quality of Service (QoS) and Quality of Context (QoC) in order to ensure that the application is using services and context information with proper QoS and QoC levels. This work presents QoMonitor, a monitoring system that assesses and monitors QoS and QoC metadata, works in both synchronous and asynchronous modes, and adopts an ontology for representing concepts in an unambiguous way. QoMonitor was integrated with OpenCOPI, a platform that facilitates the development and execution of ubiquitous applications. This work also illustrates the use of the monitor in an ubiquitous application that monitors oil wells. The results of the performance evaluation have shown that the use of the QoMonitor by OpenCOPI has not generated a considerable impact and has enabled OpenCOPI to deal with real QoS and QoC values.*

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.

Além da necessidade de se conhecer os parâmetros de qualidade dos serviços providos por diversas plataformas, as aplicações ubíquas ainda precisam lidar com as diferentes tecnologias em que as plataformas foram desenvolvidas, tornando o trabalho de desenvolver aplicações ubíquas ainda mais complexo. O OpenCOPI (*Open COntext Platform Integration*) [Lopes, 2011] é uma plataforma orientada a serviços e baseada em *workflows* semânticos que integra serviços de provisão de contexto de modo transparente e automático, provendo um ambiente que facilita o desenvolvimento e execução de aplicações sensíveis ao contexto. No entanto, o OpenCOPI não possui um módulo de monitoramento de serviços, e necessita que as próprias plataformas provedoras de serviços disponibilizem os metadados de qualidade de seus serviços. Porém, a maioria das plataformas não disponibiliza seus metadados de qualidade [Lopes, 2011], de modo que as aplicações não têm como aplicar critérios de escolha acerca de qual serviço deverá ser selecionado para ser utilizado.

Nessa perspectiva, o objetivo deste artigo é apresentar um módulo de monitoramento chamado *QoMonitor*. Esse módulo fornece uma API e afere, monitora e disponibiliza parâmetros de QoS e QoC de serviços providos por plataformas, que podem ser um *middleware* para Computação Ubíqua ou mesmo um serviço Web. Nessa perspectiva, o objetivo do *QoMonitor* é fornecer as plataformas de *middleware* de provisão de contexto, a possibilidade de fazer requisições síncronas e assíncronas para obter metadados de qualidade dos serviços. De posse desses dados, o *middleware* e suas aplicações poderá escolher qual serviço satisfaz melhor suas necessidades.

Para ilustrar a necessidade do monitoramento da qualidade de serviço e de contexto dos serviços utilizados pela aplicação, é utilizado neste trabalho um estudo de caso que consiste em uma aplicação ubíqua da área de petróleo e gás natural. Tal aplicação usa o OpenCOPI para integrar a grande quantidade de serviços necessários para realizar as suas atividades. Alguns desses serviços fornecem informações de contexto para as aplicações. O *QoMonitor* desempenha papel fundamental nesse cenário, pois diferentes serviços do estudo de caso oferecem a mesma funcionalidade, porém utilizam tecnologias diferentes o que gera diferentes metadados de qualidade para cada serviço. Esses metadados provêem ao OpenCOPI subsídios na escolha dos melhores serviços a serem utilizados.

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. A Seção 4 apresenta a avaliação do referido mecanismo de monitoramento. A Seção 5 discorre acerca de trabalhos relacionados. Por fim, a Seção 6 contém conclusões e direcionamentos para trabalhos futuros.

2. Estudo de caso: Monitoramento de poços de petróleo

O estudo de caso conduzido neste trabalho consiste em uma aplicação ubíqua na 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 o *OpenCOPI* para integrar a grande quantidade de serviços necessários para realizar as suas atividades. Alguns desses serviços fornecem informações de contexto para as aplicações e alguns oferecem a mesma funcionalidade, porém utilizam tecnologias diferentes. Nessa perspectiva, o *QoMonitor* tem papel fundamental no estudo de caso, pois provê ao *OpenCOPI* metadados de qualidade que ajudarão na escolha do melhor serviço para ser utilizado. 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 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 sobre a reconfiguração da UB, ou ainda pará-la em casos mais graves. 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. *ChangeControlSystem* armazena e recupera mudanças realizadas nos equipamentos usados na exploração de petróleo.

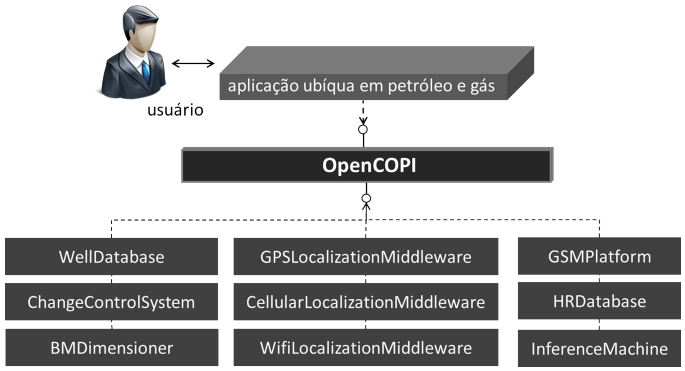

Figura 1. Provedores de serviço do estudo de caso de monitoramento de poços

Os provedores *WifiLocalizationMiddleware*, *GPSLocalizationMiddleware* e *CellularLocalizationMiddleware* são plataformas responsáveis por fornecer serviços de localização dos técnicos espalhados pelos campos de petróleo. *HRDatabase* é um sistema que fornece informação sobre os funcionários, e g. quais funcionários estão em

serviço em um dado instante. *InferenceMachine* é o provedor responsável por fazer a procura (inferências) pelas opções de regime, e selecionar um dos regimes. Por fim, *GSMPlatform* é uma plataforma que fornece um serviço de envio de mensagens SMS.

3. QoMonitor

O *QoMonitor* é um sistema de monitoramento de metadados que afere, monitora e disponibiliza metadados de qualidade de serviço (QoS) e qualidade de contexto (QoC) de serviços providos por diferentes plataformas. Para o presente trabalho foram considerados os seguintes cinco parâmetros de QoS: (i) *response time* (tempo de resposta), (ii) *MTBF* (*mean time between failures*), o tempo médio decorrido entre falhas no sistema; (iii) *MTTR* (*mean time to recovery*) o tempo médio decorrido entre uma falha no sistema e sua 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, e; (v) *uptime*, que se refere ao tempo de funcionamento (i.e. disponibilidade) de um serviço. O *QoMonitor* ainda considera alguns parâmetros de QoC: (i) *freshness* (atualidade), expressa a idade da informação, (ii) *correctness* (corretude), expressa a probabilidade de uma informação estar correta; (iii) *resolution* (resolução), descreve a granularidade das informações de contexto, e; (iv) *precision* (precisão), denota o nível de acurácia das informações de contexto.

3.1. Arquitetura

A Figura 3 ilustra a arquitetura do *QoMonitor* que foca na 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.

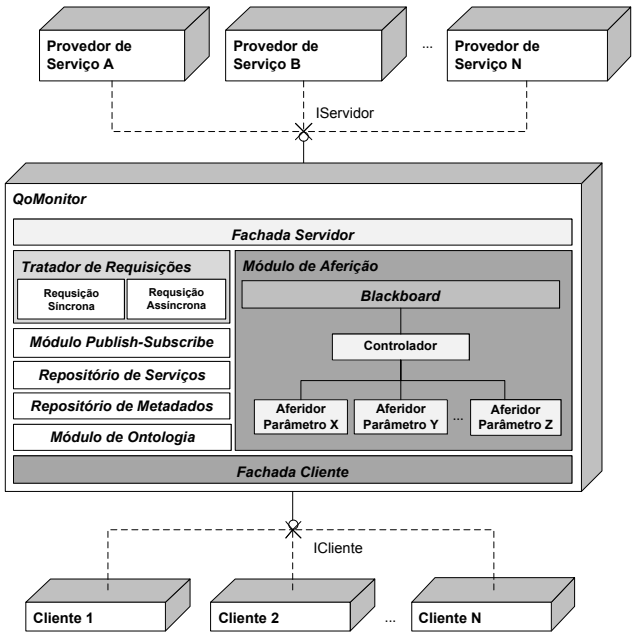

Figura 3. Arquitetura conceitual do *QoMonitor*.

A *Fachada Servidor* modulariza toda a comunicação do monitor com os provedores de serviços, sendo responsável por registrar novos serviços no *Repositório de Serviços* e se comunicar com os provedores de serviço. Por sua vez, 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 seus provedores.

A *Fachada Cliente* é responsável por fazer a comunicação do monitor com os clientes, que podem ser qualquer aplicação ubíqua ou *middleware* que necessita fazer uso do monitoramento de parâmetros de QoS e/ou QoC. Para realizar essa comunicação com os clientes, foi definida uma API (disponível em: http://consiste.dimap.ufrn.br/projects/qomonitor) que implementa a interface *ICliente*, permitindo que clientes façam requisições síncronas e assíncronas. O *Repositório de Metadados* é responsável por persistir todos os metadados dos serviços que são aferidos pelo monitor e também metadados que são disponibilizados pelos provedores de serviços. O *Módulo de Ontologia* é responsável por representar esses dados na forma de uma ontologia, conforme representado na Figura 3.

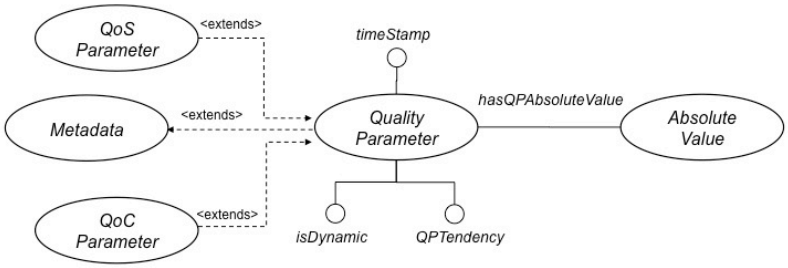

Figura 3. Ontologia empregada pelo *QoMonitor* para representação de dados

O *Módulo Publish-Subcribe* é responsável por fazer a comunicação com o cliente quando o evento da subscrição ocorre. Para realizar requisições assíncronas, os clientes devem registrar-se em um tópico JMS disponibilizado pelo monitor, executado internamente ao *Módulo Publish-Subscriber*. Quando o evento determinado pela subscrição ocorre, o resultado da subscrição é publicado no tópico, e o tópico responsabiliza-se por comunicar-se com o cliente.

Um dos principais componentes do monitor é o *Módulo de Aferição*. Esse componente é responsável por aferir os metadados de QoS/QoC dos serviços armazenados no *Repositório de Serviços* e monitorá-los, como proposto em Alves (2012), sendo composto basicamente de três tipos de componentes, a saber, *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 forneça 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* é responsável por realizar requisições periódicas aos serviços, através da *Fachada Servidor*, e armazenar as informações capturadas nessas requisições. O *Blackboard* incorpora a ideia de um repositório de dados compartilhados, 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 seja realizado, visto 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* mantém 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 fim, a ideia do componente *Controlador* é controlar o acesso às informações armazenadas no *Blackboard*, de modo que os aferidores não conhecem a origem dos dados que eles utilizam para aferição, modularizando assim a arquitetura.

O *Tratador de Requisições* é responsável por tratar as requisições síncronas e assíncronas, o que inclui o tratamento da condição de retorno. Para tratar as requisições, o *Tratador de Requisições* recupera os dados de QoS/QoC através do *Módulo de Ontologia* e os repassa para os clientes que estão solicitando os referidos dados através da *Fachada Cliente*. Quando uma requisição assíncrona é realizada, uma condição de retorno é passada pelo cliente para indicar quando o cliente deve ser notificado. Para tratar a condição de retorno o tratador faz uso de um interpretador (*parser*) apresentado no trabalho Alves (2012).

3.1. Funcionamento

3.1.1. Requisição Síncrona

Para realizar uma requisição sínrconra, é necessário que o cliente invoque o método *getServiceQuality* da API do *QoMonitor* e passe como parâmetros os dados do serviço monitorado (nome, endereço IP, porta, e lista de parâmetros de entrada) e a lista de parâmetros de qualidade a ser enviada ao cliente. Por exemplo, utilizando o serviço Web de envio de mensagens SMS do estudo de caso deste trabalho, o formato da requisição síncrona seria: <*GSMPlatform.SendSMStoEmployes*, *192.168.1.100*, *8080*, *Cellphone*, *Message*> (dados do serviço monitorado), <*ResponseTime*, *ErrorRate*, *MTBF*> (lista de parâmetros). Depois que a requisição é tratada, o resultado é repassado à *Fachada Cliente*, que responde ao cliente. Um exemplo de retorno seria: <*ResponseTime* = 10.1 ms; *ErrorRate* = 0%; *MTBF* = 1 s >. Um detalhe a ser observado é que o tempo de resposta das requisições síncronas é apenas o tempo de rede somado ao tempo de processamento das informações. No entanto, caso o cliente faça uma requisição passando dados de um serviço que ainda não está sendo monitorado, o monitor comporta-se de forma diferente e o tempo de resposta da requisição pode aumentar devido os metadados de qualidade do serviço não estarem no *Repositório de Metadados*, forçando o monitor a iniciar imediatamente o monitoramento e aferição dos parâmetros de qualidade do serviço.

Outra forma de realizar uma requisição síncrona é chamando o método *getServicesQuality*. O método possui a mesma execução do método *getServiceQuality*, mas em vez de receber como parâmetro de entrada dados de um único serviço, o método recebe uma lista de dados de vários serviços e retorna uma outra lista com os metadados desses vários serviços. Esse método permite que clientes recuperem metadados de serviços diferentes com uma única requisição.

3.1.1. Requisição Assíncrona

Para realizar uma requisição assíncrona, o cliente primeiro subscreve-se para cadastrar o interesse em receber notificações sobre paramentos de QoS ou QoC

monitorados pelo *QoMonitor* invocando o método *subscribeServiceQuality* da API do *QoMonitor* que recebe como parâmetros: (i) os dados do serviço monitorado (nome, endereço IP, porta e lista de parâmetros de entrada); (ii) uma lista de serviços e os parâmetros a serem enviados ao cliente, e; (iii) uma determinada condição de retorno formada pela tupla <*expressão*, *tipo de comparação*, *valor*>, de modo que o retorno desse tipo de requisição só é realizado quando tal condição de retorno é satisfeita. Por exemplo, utilizando o serviço Web de envio de mensagens SMS e o serviço de localização por GPS do estudo de caso desse trabalho, o formato da requisição assíncrona seria: (i) dados do serviço monitorado: <<*GPSLocalizationMiddleware*, *192.168.1.100*, *8080*, *Field*, *OilWell*>; <*GSMPlatform*, *192.168.1.100*, *8080*, *Cellphone*, *Message*>>; (ii) lista de serviços e seus parâmetros: <*GSMPlatform*<*ResponseTime*, *ErrorRate*, *MTBF*>; *GPSLocalizationMiddleware*<*ResponseTime*, *ErrorRate*, *MTTR*>>; (iii) condição de retorno: <(#{*GSMPlatform.MTTR*} + #{*GPSLocalizationMiddleware.MTTR*}), *igual*, *10*>. Quando um cliente realiza uma subscrição ele imediatamente recebe um identificador, que é um valor numérico que identifica a subscrição no monitor. Caso esse valor seja negativo, é indicado ao cliente que algum parâmetro que ele passou está incorreto.

O primeiro parâmetro da tupla que compõe a condução de retorno é uma *expressão* formada por variáveis e operações matemáticas determinadas pelo usuário que, aplicadas aos dados armazenados pelo monitor, retornam um valor. A ideia de tratar com expressões surgiu da necessidade dos clientes serem avisados quando um conjunto de eventos ocorresse, e não somente um. Através dessa *expressão*, o cliente pode utilizar todos os parâmetros de todos os serviços tratados pelo monitor como variáveis e utilizar várias operações matemáticas disponíveis. A *expressão* é tratada por um componente chamado *Parser* presente dentro do *Tratador de Requisições*. Caso a equação esteja incorreta, o cliente recebe um valor negativo, indicando que a subscrição não foi realizada com sucesso. Já o parâmetro *valor* da tupla é um valor numérico qualquer determinado pelo usuário. Por fim, o parâmetro *tipo de comparação* determina como será comparado o valor retornado pela *equação* com o *valor* passado pelo cliente. Os tipos de comparação possíveis são: (i) *maior*; (ii) *menor*; (iii) *igual*; (iv) *maiorIgual*, e; (v) *menorIgual*. Por exemplo, quando o MTTR somado ao MTBF, ambos parâmetros representados em segundos, do serviço *GSMPlatform* do estudo de caso, ultrapassasse o valor *10*, a condição de retorno ficaria: <#{*GSMPlatform.MTTR*} + #{*GSMPlatform.MTBF*}, *maior*, *10*>. De posse dos dados da requisição, o *Tratador de Requisições Assíncronas* monitora continuamente os dados a fim de identificar se a condição de retorno passou a ser satisfeita. Quando satisfeita, o tratador responde imediatamente ao *Módulo Publish-Subscribe* que publica no tópico JMS e esse tópico trata de responder ao cliente. Nesse caso, foi determinado que, para realizar subscrições, o cliente precisa registrar-se em um tópico JMS disponibilizado pelo monitor. Para se registrar no tópico JMS o cliente repassa o identificador da subscrição.

4. Avaliação

A avaliação quantitativa teve como objetivo endereçar, através de experimentos computacionais, o tempo despendido pelo monitor para responder a requisições síncronas feitas pelo OpenCOPI e o *overhead* causado pelo uso do monitor. A avaliação que aborda o cenário assíncrono é apresentada em Alves (2012). Através dos metadados de qualidade fornecidos pelo *QoMonitor*, o OpenCOPI selecionará qual o melhor plano

de execução. A partir dos serviços disponíveis foi possível criar cinco diferentes configurações de workflows para a aplicação, variando de 2 a 32 planos de execução (descrição completa das configurações e figuras dos planos de execução disponíveis em: http://consiste.dimap.ufrn.br/projects/qomonitor/ctic2013). Para cada uma das configurações, foram feitas 20 execuções independentes do processo de seleção realizado pelo OpenCOPI, que desconsidera metadados de serviços coincidentes, ou seja, serviços que estão presentes em todos os planos de execução e, portanto, contribuem igualmente nos cálculos do processo de seleção [Cavalcante et al., 2012]. Com isso, o OpenCOPI só necessita recuperar metadados do QoMonitor acerca dos serviços que não são coincidentes.

O OpenCOPI foi executado em um computador com processador Intel® Core™ i7 2.10 GHz, 8 GB de memória RAM e sistema operacional Linux Ubuntu versão 12.10, e realizava requisições síncronas ao monitor sempre que executava o processo de seleção de um plano de execução. Nos experimentos computacionais, o *QoMonitor*, implantado no servidor de aplicação Apache Tomcat instalado em um computador com processador Intel® Core™ i5 2.3 GHz, 4 GB de memória RAM e sistema operacional Mac OS X 10.8.2, respondia às requisições síncronas do OpenCOPI, retornando uma lista de metadados de qualidade dos serviços não coincidentes. De forma a realizar tais experimentos em um ambiente controlado, o *QoMonitor* e o OpenCOPI foram colocados em redes locais cabeadas, de modo a minimizar a influência da rede no processo.

A Figura 4 apresenta, nesta ordem, a média, das 20 execuções realizadas, dos tempos (em milissegundos) de resposta à requisição síncrona do *QoMonitor* (*Rec. Met. QoM*), o tempo para recuperação dos metadados pelo OpenCOPI (*Rec. Metadados*), o tempo total da seleção (*Seleção*) e o tempo despendido para o processo de seleção propriamente dito (*Apenas seleção*), desconsiderando a recuperação dos metadados. O tempo de resposta da requisição síncrona corresponde à diferença entre o instante de tempo no qual a requisição chega ao *QoMonitor* e o instante no qual a requisição é respondida. O tempo para a recuperação dos metadados de qualidade é basicamente o tempo despendido pelo OpenCOPI para realizar a requisição síncrona, usando o método *getServicesQuality*, e receber os metadados dos serviços passados como parâmetro. O tempo total da seleção corresponde ao tempo despendido pelo OpenCOPI para realizar a seleção do melhor plano de execução, levando em consideração os valores dos metadados de qualidade recuperados através do *QoMonitor*. Por fim, o tempo do processo de seleção propriamente dito corresponde ao tempo que o OpenCOPI leva para selecionar o plano de execução, desconsiderando a recuperação dos metadados. A tabela com todos os valores utilizados para gerar o gráfico da Figura 4 está disponível em: http://consiste.dimap.ufrn.br/projects/qomonitor/ctic2013.

Pode-se observar que boa parte do tempo para receber uma resposta de uma requisição síncrona é influenciada pela rede. Por exemplo, para responder a uma requisição síncrona com 2 serviços, o *QoMonitor* levou em média 9,1 ms, enquanto para receber a resposta dessa mesma requisição o OpenCOPI teve que esperar 34,46 ms em média, ou seja, o tempo da rede foi de aproximadamente 25 ms. É possível observar também que boa parte do tempo para seleção corresponde ao tempo de recuperação dos metadados através da requisição síncrona, e apesar disso nenhuma requisição levou mais de 200 ms para ser respondida, ou seja, o uso do monitor não promove um impacto considerável no OpenCOPI. É possível observar na Figura 4 que com o aumento dos

planos de execução há um aumento no tempo para recuperação dos metadados. Outro detalhe importante é que, com as requisições síncronas, o OpenCOPI pode tratar com valores de QoS e QoC reais e não mais dados fictícios, o que gera uma grande vantagem em utilizar o *QoMonitor*.

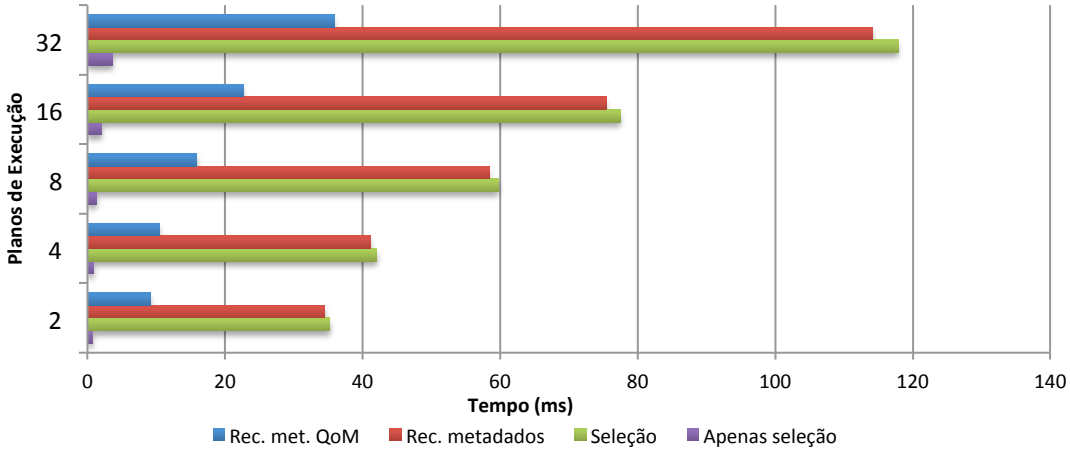

Figura 4 - Gráfico mostrando evolução da média dos tempos

5. Trabalhos relacionados

Os trabalhos relacionados possuem enfoque no uso de dados de qualidade obtidos a partir do monitoramento de sistemas distribuídos. Huebscher *et al*. (2005) apresentam um *framework* que abstrai das aplicações sensíveis a contexto informações relacionadas à origem das informações obtidas. O *framework* possui uma camada chamada *Context Service*, responsável por escolher qual provedor de contexto que melhor satisfaz os valores de QoC determinados pela aplicação cliente, dentre os diversos que disponibilizam a mesma informação de contexto, e abstrair a comunicação das aplicações com o provedor de contexto escolhido. A camada *Context Service* ainda mantém um monitoramento constante dos sensores para que seja possível trocar de provedor de contexto em tempo de execução quando um novo provedor de contexto aparece, quando um provedor falha, ou quando há degradação da qualidade de contexto. As aplicações podem determinar pesos para os atributos de qualidade. Todavia, o trabalho de Huebscher *et al*. (2005) restringe-se a tratar com apenas dois metadados de *QoC* e não permite subscrições das aplicações sensíveis a contexto, além de não apresentar mecanismos de aferição dos metadados de *QoC*, caso o provedor de contexto não disponibilize esses metadados. Zheng *et al*. (2011) apresenta um *framework* sensível a contexto que utiliza parâmetros de *QoC*, definidos pelas aplicações clientes, para selecionar provedores de contexto. O *framework* captura e interpreta informações obtidas de provedores de contexto para que sejam usadas por aplicações clientes. Antes de repassar as informações para os clientes, o *framework* realiza vários processamentos que funcionam como filtros para descartar informações de contexto duplicadas, inconsistentes ou que não cumprem os níveis de *QoC* necessários. O *framework* possui uma ontologia para representação das informações de contexto e outra para representação dos parâmetros de *QoC*. Apesar desse *framework* também definir uma ontologia para representação dos parâmetros de QoC, possuir uma arquitetura modularizada e tratar com mais parâmetros de QoC do que o monitor desenvolvido neste trabalho, o *framework* não apresenta mecanismos para tratar requisições síncronas e subscrições de aplicações clientes, nem apresenta mecanismos de aferição dos

metadados de QoC. Tian *et al*. (2004) apresentam uma abordagem que possibilita o monitoramento e seleção baseada em QoS de serviços Web através dos seguintes componentes: (i) *WS-QoS Editor*; (ii) *WS-QoS Requirement Manager*; (iii) *WS-QoS Broker*, e; (iv) *WS-QoS Monitor*. *O WS-QoS Editor* permite que os clientes possam editar as suas preferências de QoS e os parâmetros de QoS. Os valores de QoS ficam armazenados estaticamente em arquivos XML. O *WS-QoS Requirement Manager* captura os dados armazenados nos arquivos XML para transformá-los no formato da ontologia. O *WS-QoS Broker* realiza a comunicação dos clientes com os serviços Web e também funciona como *cache* armazenando dados de serviços Web que foram requeridos recentemente. Finalmente, o *WS-QoS Monitor* exibe os parâmetros de QoS oferecidos pelos serviços Web e as preferências de QoS dos clientes, para que seja verificado se os valores de QoS oferecidos atendem ao que está sendo requisitado. Esse trabalho, apesar de definir uma ontologia extensível para representação dos parâmetros de QoS que permite a definição de novos parâmetros, não apresenta mecanismos para tratar requisições síncronas e subscrições de aplicações clientes, nem mecanismos de aferição dos metadados de QoS, caso o serviço Web não os disponibilize.

6. Conclusão

Nesse trabalho foi apresentado o *QoMonitor*, um sistema de monitoramento de metadados que funciona como um serviço Web e que afere, monitora e disponibiliza, através de uma API, metadados de qualidade de serviço (QoS) e qualidade de contexto (QoC) de serviços providos por diferentes plataformas. Os metadados de qualidade aferidos são representados através de uma ontologia, com o objetivo de prevenir ambiguidade de interpretações dos dados e permitir reuso, extensões e até traduções para outras ontologias. Nos experimentos realizados para avaliar o *QoMonitor* foi possível observar que o seu uso no contexto do OpenCOPI não gerou um impacto considerável e possibilitou ao OpenCOPI trabalhar com valores reais de QoS e QoC.

Referências

Dey, A.; Abowd, G.; Salber, D. (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.

Weiser, M. (1991) "The computer of the Twenty-First Century". Scientific American, vol. 265, no. 3, pp.94–104.

Cavalcante, E. et al. (2012) "Optimizing services selection in a cloud multiplatform scenario". Proc. of the 2012 IEEE Latin America Conf. on Cloud Computing and Communications, pp. 31-36.

Huebscher, M.; McCann, J. (2005) "An adaptive middleware framework for context-aware applications". Personal and Ubiquitous Computing, vol. 10, no. 1, pp.12–20.

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

Tian, M.; Gramm, A.; Ritter , H.; Schiller, J. (2004) Efficient Selection and Monitoring of QoS-Aware Web Services with the WS-QoS Framework, Proc. of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence, pp.152-158

Alves, G (2013) Um sistema para monitoramento de metadados para Computação Ubíqua. Monografia de Graduação em Ciência da Computação – UFRN, Natal, Brasil.

Lopes, F (2011) Uma plataforma de integração de middleware para Computação Ubíqua. Tese de Doutorado (Doutorado em Ciência da Computação) – Programa de Pós-Graduação em Sistemas e Computação, UFRN, Natal, Brasil.