5
Avaliações

Em nossa avaliação procuramos verificar, em primeiro lugar, se o modelo de dados é capaz de oferecer respostas para perguntas envolvendo aspectos do sistema que permitem, ao desenvolvedor, melhor compreender o comportamento do sistema. Para obter tais perguntas, pedimos aos desenvolvedores que listassem perguntas que pudessem guiá-los durante o processo de diagnóstico e depuração de um dado problema.

Buscamos avaliar também o impacto que o nosso mecanismo impõe no desempenho da aplicação monitorada. Com essa avaliação procuramos oferecer uma estimativa da sobrecarga, de modo que o desenvolvedor ou administrador possa avaliar se a redução no desempenho é tolerável ou não para um caso especifico.

Para os experimentos deste capítulo utilizamos uma aplicação que reproduz a dinâmica das interações que ocorrem em um ambiente real de produção, porém evitando processar o imenso volume de dados utilizados nesse ambiente. Essa simplificação é aceitável e não influi na análise feita nesse trabalho, pois o foco de nossa análise são as interações entre os componentes, que foram preservadas. O processamento dos dados apenas aumentaria o tempo de execução dos métodos invocados, mas não altera as sequências de interações entre os componentes da aplicação.

Descreveremos, na seção 5.1 deste capitulo, a aplicação utilizada nos experimentos e em seguida (seção 5.2) apresentaremos a avaliação do nosso modelo de dados e os testes realizados para medir a sobrecarga causada na aplicação analisada pelo interceptador de instrumentação.

5.1
Aplicação utilizada

Para validar este trabalho utilizamos nossa ferramenta em uma aplicação do *Openbus* [21], um *middleware* para integração de aplicações baseadas em componentes distribuídos que é utilizado em um ambiente real de produção. Esse *middleware* utiliza o padrão CORBA para permitir a comunicação de aplicações desenvolvidas em diferentes linguagens. Essa comunicação é feita

através de um barramento onde as aplicações se conectam para oferecer ou utilizar serviços. A conexão é feita através de uma autenticação feita por meio do par usuário-senha ou por meio de um certificado digital que segue a especificação X.509 (RFC 3280 [22]) da IETF (*Internet Engineering Task Force*).

O *Openbus* é composto de três serviços básicos (figura 5.1) que gerenciam as conexões ao barramento e uma biblioteca (específica para cada linguagem) para o desenvolvimento de serviços que devam ser integrados utilizando o *Openbus*. Os serviços básicos são o *AccessControlService*, para controle do acesso ao barramento, o *RegistryService*, para o gerenciamento de ofertas de serviços e *SessionService*, para gerenciamento de sessões.

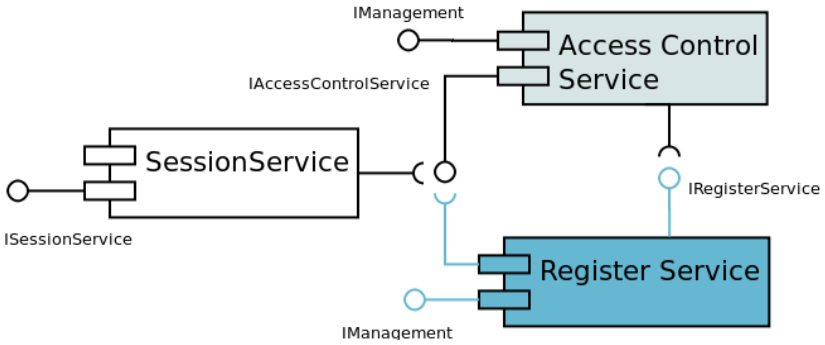

Figura 5.1: Openbus: Serviços Básicos

Para que uma dada aplicação tenha acesso ao barramento, ela deve fazer uma conexão, utilizando o serviço básico para controle de acesso, e obter uma credencial que será usada sempre que a aplicação acessar o barramento. Se a aplicação tiver algum serviço a ser oferecido, ela vai registrar uma oferta de serviço, explicitando as interfaces que são implementadas por este. Já no caso de uma aplicação cliente, ela vai procurar uma oferta de serviço referente a uma dada interface e então poderá utilizar um *proxy* para acesso ao serviço ofertado. Essa busca por ofertas de serviços que implementem uma determinada interface é feita através do serviço de registro. Esse serviço permite, também, que uma aplicação registre uma oferta de serviço para uma ou mais interfaces. A credencial usada para acesso ao barramento é válida por um dado período de tempo, que é configurável, devendo ser renovada antes do decurso deste período.

A aplicação utilizada nos experimentos deste trabalho (figura 5.2) utiliza esse barramento de comunicação. Nosso objetivo é utilizar a nossa ferramenta para analisar as interações que ocorrem nessa aplicação, a fim de obter informações para auxiliar os desenvolvedores no diagnóstico das causas de comportamentos inesperados ou de falhas no sistema.

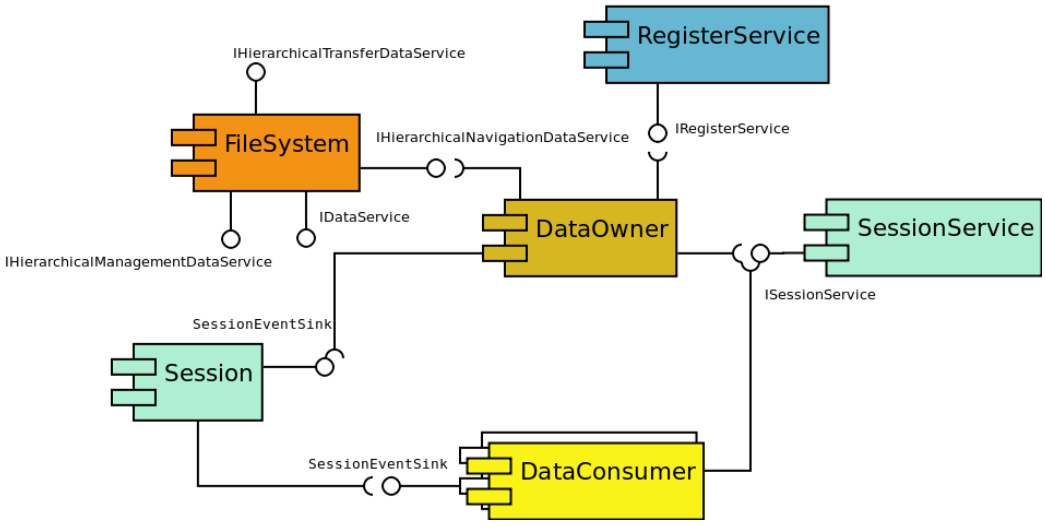

Figura 5.2: Diagrama de componentes da aplicação

Essa aplicação é composta por um serviço baseado na arquitetura para acesso a dados estruturados em aplicações cientificas. Essa arquitetura, proposta por Henrique [23], permite o compartilhamento de dados entre aplicações dessa natureza, que manipulam grandes volumes de dados. No exemplo usado neste trabalho, a aplicação possui um serviço que permite o acesso remoto ao sistema de arquivos (componente *FileSystem*). Esse serviço é utilizado pelo componente *DataOwner* para recuperar um dado específico que será utilizado por um terceiro serviço, um consumidor. A comunicação entre esses dois serviços é feita através de uma sessão criada, representada por um componente *Session*.

Como cenário de testes, foi criada uma estrutura de diretórios com arquivos de dados e os serviços básicos do barramento foram inicializados. Em seguida, foram inicializados o componente *FileSystem*, que se registrou nesse barramento, e o componente *DataOwner*, que se conectou ao barramento e criou uma sessão para comunicação. Por fim, foi inicializado o consumidor, que se adicionou como membro dessa sessão. Os testes foram realizados com 100 iterações do componente *DataOwner* que obtinha os dados do componente *FileSystem* e os passava para o consumidor utilizando a sessão criada para a comunicação entre eles.

Como será visto nas próximas seções, com a nossa ferramenta foi possível verificar alguns aspectos do sistema, como a sequência de interações realizadas pelos componentes para conexão ao barramento e para desconexão. Foi possível também observar que os serviços conectados realizavam periodicamente uma chamada remota para renovação da credencial obtida para utilização do

barramento.

Uma verificação inicial, bem simples, que pôde ser feita com a nossa ferramenta, foi identificar quando um serviço deixava de executar o método para renovação da credencial (`renewLease`), o que pode ser um indicativo de algum problema no serviço, já que o comportamento esperado é a execução periódica desse método.

Esse comportamento se refletia em atualizações periódicas do visualizador para exibir a nova transação que incluía aquele método. A ausência de qualquer atualização por um período de tempos maior que alguns minutos, indica a necessidade de verificar se o serviço ainda está ativo.

5.2 Avaliação do modelo de dados

Essa avaliação foi feita utilizando o *Openbus* como estudo de caso. Para isso buscamos junto aos desenvolvedores de serviços que utilizam o barramento qual o tipo de informação eles julgavam necessárias ou convenientes para auxiliar a análise do sistema e facilitar a depuração de um comportamento anômalo no sistema. Assim, obtivemos dados que nos propiciaram gerar relatórios que elucidavam aspectos do sistema relevantes do ponto de vista dos desenvolvedores. Foi possível também analisar o quanto o nosso modelo de dados conseguia responder questões objetivas que influenciam a análise por parte do desenvolvedor e do administrador do sistema.

As questões levantadas pelos desenvolvedores estão enumeradas a seguir:

1. Qual o tempo de funcionamento do barramento (*uptime*).
2. Relatório de publicações de serviços realizadas em um período de tempo indicado pelo usuário.
3. Quais os serviços que estão publicados no barramento em um dado intervalo de tempo.
4. Lista de logins realizados em um período de tempo definido pelo usuário.
5. Lista de logins realizados por um serviço em um determinado intervalo de tempo.
6. Lista de logins que falharam.
7. Lista de transações nas quais ocorreu uma exceção.
8. Lista de credenciais válidas em um determinado instante.

Todas as questões levantadas foram analisadas a fim de se identificar como essas questões se relacionavam com o nosso modelo de dados. Nessa análise, visamos determinar uma forma de mapear as questões apresentadas em consultas à nossa base de dados, feitas em linguagem SQL. Quase todos os aspectos levantados puderam ser mapeados em uma consulta que permitia responder a questão proposta, com exceção do item 6. Esse fato mostrou uma limitação da nossa ferramenta que não relaciona as transações com eventos específicos da aplicação, como discutiremos mais adiante. As consultas SQL são apresentadas nas listagens 5.1 e 5.2.

codigo 5.1: Consultas SQL

```
1  --Lista de logins realizados em um periodo
2  --de tempo definido pelo usuário.
3  --parametros tempoIni, tempoFinal
4
5  SELECT t.timestamp, t.User, TransactionID, OriginContainer, DestContainer,
6      DestMethod
7  FROM Transactions as t, RemoteCall
8  WHERE (DestMethod LIKE 'loginBy%')
9  AND (DestContainer = 'AccessControlService')
10 AND (t.rowid = RemoteCall.TransactionID)
11 AND (t.timestamp BETWEEN tempoIni AND tempoFinal)
12 ORDER BY t.timestamp
13
14 --Lista de logins realizados por um serviço em
15 --um determinado intervalo de tempo.
16 --parametros nomeDoServiço, tempoIni, tempoFinal
17
18 SELECT t.timestamp, t.User, TransactionID, OriginContainer, DestContainer,
19     DestMethod
20 FROM Transactions as t, RemoteCall
21 WHERE (DestMethod LIKE 'loginBy%')
22 AND (DestContainer = 'AccessControlService')
23 AND (t.rowid = RemoteCall.TransactionID)
24 AND (OriginContainer LIKE nomeDoServiço)
25 AND (t.timestamp BETWEEN tempoIni AND tempoFinal)
26 ORDER BY t.timestamp
27
28 --Lista de logins realizados por um usuário em
29 --um determinado intervalo de tempo
30 --parametros nomeDoUsuário, tempoIni, tempoFinal
31
32 SELECT t.timestamp, t.User, TransactionID, OriginContainer,
33         DestContainer DestMethod
34 FROM Transactions as t, RemoteCall WHERE (DestMethod LIKE 'loginBy%')
35 AND (t.rowid = RemoteCall.TransactionID)
36 AND (DestContainer = 'AccessControlService')
37 AND (t.User LIKE 'tester')
38 AND (t.timestamp BETWEEN tempoInicial AND  tempoFinal)
39 ORDER BY t.timestamp
40
41 --Lista de publicações de serviços realizadas em
42 --período de tempo indicado pelo usuário
43 --parametros: tempoInicial e tempoFinal
44
45 SELECT t.timestamp, t.rowid, OriginContainer, OriginComponent,
46         DestContainer, DestComponent, DestMethod
47 FROM Transactions as t, RemoteCall
48 WHERE (DestMethod like 'register') AND (DestContainer = 'RegistryService')
49 AND (t.rowid = RemoteCall.TransactionID)
50 AND (t.timestamp BETWEEN tempoInicial AND tempoFinal)
51 ORDER BY t.timestamp
```

codigo 5.2: Consultas SQL

```
1 --Lista de serviços publicados no barramento em um determinado instante
2 --Serviço que realizaram o register mais não realizaram nenhum
3 --unregistry posterior ao Register
4
5 SELECT t.timestamp, t.rowid, OriginContainer, OriginComponent,
6        DestContainer, DestComponent, DestMethod
7 FROM Transactions as t, RemoteCall
8 WHERE (DestMethod like 'register') AND (DestContainer = 'RegistryService')
9 AND (t.rowid = RemoteCall.TransactionID)
10 AND (t.timestamp BETWEEN tempoIni AND tempoFinal)
11 AND (OriginComponent NOT IN (
12     SELECT OriginComponent FROM RemoteCall as r WHERE
13     (r.DestMethod like 'unregister') AND (r.DestContainer = 'RegistryService')
14     AND r.sReqTimestamp > RemoteCall.rReqTimestamp))
15 ORDER BY t.timestamp
16
17 --Uptime do barramento.
18
19 SELECT r.sReqTimestamp FROM RemoteCall as r
20 WHERE (DestContainer like 'AccessControlService%')
21 AND (DestMethod = 'renewLease')
22 AND (OriginComponent like 'RegistryService')
23 AND (r.sReqTimestamp) >= (strftime('%s','now') - tempoDeValidadeDoLease)
24
25 --Hora do último login feito pelo RegistryService
26 SELECT MAX(r.sReqTimestamp) FROM RemoteCall as r
27 WHERE (DestContainer like 'AccessControlService%')
28 AND (DestMethod like 'loginBy%')
29 AND (OriginComponent like 'RegistryService')
30
31
32 --Hora do último renewLease do RegistryService registrado
33 SELECT MAX(r.sReqTimestamp) FROM RemoteCall as r
34 WHERE (DestContainer like 'AccessControlService%')
35 AND (DestMethod = 'renewLease')
36 AND (OriginComponent like 'RegistryService')
37 AND (r.sReqTimestamp) >= (strftime('%s','now') - tempoDeValidadeDaCredencial)
38
39
40 --Lista de credenciais válidas em um determinado instante dado por
41 --Parametros: timeInstance, tempoDeValidadeDaCredencial
42
43 SELECT r.originContainer, t.user, t.credential
44 FROM RemoteCall as r, Transactions as t
45 WHERE (DestContainer like 'AccessControlService%')
46 AND (DestMethod = 'renewLease')
47 AND (r.sReqTimestamp) >= (timeInstance - tempoDeValidadeDaCredencial)
48 AND r.TransactionID = t.id
```

Para responder a questão do item 1 vamos considerar que o barramento está disponível somente caso os dois serviços básicos (o *RegisterService* e o *AccessControlService*) estejam executando. Nesse caso, o *RegisterService* deve fazer chamadas periódicas para renovar a credencial. Então, caso tenha passado o tempo de validade da credencial sem que exista uma chamada remota para sua renovação, poderemos afirmar que o *RegisterService* não possui mais uma credencial válida e vamos considerar que o barramento não está disponível. O tempo de *uptime* do barramento vai ser o tempo decorrido desde o login do *RegisterService* até o momento de sua desconexão (chamada ao método `logout`), ou até que tenha se passado um tempo maior que a validade da credencial, sem que o serviço a tenha renovado (ou seja, sem que tenha feito

uma chamada ao método `RenewLease`).

Para que um serviço seja publicado no barramento é necessário que esse faça o registro de uma oferta de serviço junto ao *RegisterService*, utilizando o método `register`. Então, para responder a questão levantada no item 2 devemos obter uma lista dos serviços que invocaram o método `register` do *RegisterService*, dentro do intervalo de tempo considerado. Já para o item 3 a lista deve se restringir ao serviços que não tenham invocado o método `unregister` dentro do intervalo de tempo considerado.

No caso do item 4, devemos obter as transações que incluem uma chamada ao método `loginByCertificate` ou `loginByPassword` e listar os usuários associados àquela transação. A questão no item 5 é análoga, mas devemos listar o serviço responsável pela transação.

No item 7 devemos obter uma lista dos métodos que retornaram com exceção. A informação sobre a sequência de métodos que levou a manifestação da exceção vai ajudar o desenvolvedor a localizar a mensagem de erro nos *logs* da aplicação.

Por fim, o item 8 estará satisfeito se obtivermos as credenciais de todos os serviços que não realizaram uma chamada ao método `renewLease` por um tempo maior que o tempo de validade da credencial. Ou seja, devemos encontrar os serviços que realizaram uma chamada a esse método no intervalo entre o instante de tempo t atual e o instante de tempo $(t - tempoValidadeCredencial)$.

Não foi possível obter a lista de *logins* que falharam (item 6) uma vez que a falha na tentativa de conexão ao barramento é reportada através de um valor booleano retornado pelas funções utilizadas para *login* (`loginByCertificate` e `loginByPassword`).

Isso ocorreu em razão de uma limitação existente no mapeamento CORBA para Java que impossibilita o acesso dos interceptadores aos valores de retorno e os parâmetros das funções invocadas. Na impossibilidade de acesso ao valor de retorno da função, seria necessário, para identificar uma falha na conexão, que a nossa ferramenta pudesse correlacionar os dados das transações com eventos da aplicação, de modo que uma chamada a função `loginByCertificate` pudesse ser correlacionada com o *log* da aplicação onde a falha foi reportada.

Devemos ressaltar, no entanto, que essa limitação é especifica da linguagem java e que não ocorreria, por exemplo em C++ ou Lua. Optamos por registar a necessidade de melhoramento da nossa ferramenta para que a nossa solução possa ser usada do mesmo modo, quer em Java, quer em Lua ou C++.

5.3 Impacto no desempenho

Nos testes de desempenho obtivemos a média do tempos total de execução e do tempo de resposta de alguns métodos dos serviços básicos do *Openbus* e dos componentes da aplicação utilizada em nossa avaliação (sessão 5.1). Nosso objetivo foi o de comparar essa média em duas situações: com e sem a utilização da instrumentação.

Para obtenção dessas médias, os métodos foram executados 200 vezes em cada situação. A partir da comparação entre a média obtida sem o uso da instrumentação com a média obtida utilizando a instrumentação, foi possível avaliar a sobrecarga introduzida pela instrumentação na execução dos métodos remotos. Todos os testes foram realizados em um computador com processador Intel Core 2 Duo T8100/2.1 GHz com 4 gigas de memória RAM e Interface de rede Ethernet.

Método	**Tempo total (ms)**	
	Instrumentado	**Não instrumentado**
register	125.81	91.83
removeMember	617.68	588.13
unregister	48.08	19.56
logout	31.81	3.52
getChallenge	32.08	3.58
loginByCertificate	55.18	21.22
findByCriteria	28.65	3.25
find	32.65	3.32
getIdentifier	23.08	3.12
createSession	1373.86	1350.69
getSession	27.07	3.83
getMembers	26.31	3.51
addObserver	76.86	50.81
addCredentialToObserver	36.18	3.78
removeCredentialFromObserver	33.57	3.81
renewLease	47.83	8.63
isValid	58.48	24.57
getSystemDeployment	31.01	3.38
areValid	134.40	104.64
_non_existent	52.70	27.62

Tabela 5.1: Tabela com os valores médios dos tempos totais de execução dos métodos

A tabela 5.1 mostra a sobrecarga medida no tempo total de execução de alguns dos métodos remotos dos serviços básicos do *Openbus*. Devemos observar que o tempo total de execução é medido pelo cliente e portanto está

sujeito a variações da rede, pois o tempo de comunicação de rede pode variar em cada invocação remota.

Nos serviços básicos do *Openbus*, implementados em linguagem Lua, o uso da instrumentação acarreta um aumento médio de 28,56 milissegundos no tempo execução dos método remotos. O percentual dessa sobrecarga em cada método é tanto maior quanto menor for o tempo médio de execução daquele método. Esse percentual é calculado tomando-se a diferença entre os tempos de execução medidos com instrumentação (tme_i) e aqueles medidos sem a instrumentação (tme), dividida pelo tempo de execução medido sem instrumentação. Portanto, quanto menor o tempo médio de execução de um método, maior será o valor percentual da sobrecarga devido a instrumentação.

$$sobrecarga(\%) = \frac{tme_i - tme}{tme}$$

Por exemplo se considerarmos o método `removeMember` (tabela 5.1), do serviço de sessão (*SessionService*), podemos verificar que a sobrecarga é de apenas 5.02%, já que á média do tempo total de execução é grande se comparado aos 28,56 milissegundos introduzido pela instrumentação. No entanto, na mesma tabela vemos que para o método `find` do serviço de registro, a sobrecarga é de mais de 800%. Isso porque o tempo médio de execução é pequeno se comparado ao tempo adicional introduzido pelo mecanismo de acompanhamento de transações.

Os valores da sobrecarga considerando apenas o tempo de reposta dos métodos remotos, medidos no servidor, estão apresentados nas tabelas 5.3 e 5.4. Nesse caso a sobrecarga foi ligeiramente menor dado que, na sobrecarga medida no tempo de resposta pesa apenas parte da sobrecarga imposta pelo nosso mecanismo de acompanhamento das transações. Essa parte é aquela relativa aos pontos de interceptação *ReceiveRequest* e *SendReply*, ou seja, relativa ao interceptador servidor.

A seguir, vamos apresentar alguns gráficos contrastando os tempos medidos sem a instrumentação com os tempos medidos após a introdução da instrumentação. Os gráficos das figuras 5.3 e 5.4 mostram a comparação entre os tempos totais de execução para os métodos dos serviços básicos do *Openbus*. Os tempos totais para os métodos do serviço de dados (*FileSystem*), implementado em linguagem Java, estão nos gráficos das figuras 5.5 e 5.6. Em todos os gráficos o intervalo de confiança é de 95%.

A comparação levando em consideração os tempos de resposta pode ser vista nos gráficos das figuras 5.7 e 5.8, para os serviços básicos do *Openbus*. A mesma comparação para os métodos do serviço de dados, está ilustrada nos gráficos das figuras 5.9 e 5.10.

Método	**sobrecarga**	
	Diferença (ms)	**Percentual**
register	33.97	36.99%
removeMember	29.55	5.02%
unregister	28.52	145.81%
logout	28.29	802.51%
getChallenge	28.50	796.78%
loginByCertificate	33.96	160.05%
findByCriteria	25.40	780.54%
find	29.34	884.77%
getIdentifier	20.97	1979.31%
createSession	23.17	1.72%
getSession	23.24	606.57%
getMembers	22.80	649.93%
addObserver	26.05	51.26%
addCredentialToObserver	32.40	856.93%
removeCredentialFromObserver	29.77	781.61%
renewLease	39.19	453.99%
isValid	33.91	138.03%
getSystemDeployment	27.63	818.70%
areValid	29.76	28.45%
_non_existent	25.08	90.81%

Tabela 5.2: Tabela com os valores médios da sobrecarga nos tempos totais de execução

Considerando os valores obtidos vemos que nos testes feitos em linguagem Java, o mecanismo de transações acarretou um aumento médio de 24,58 milissegundos no tempo total de execução de cada método. Já em Lua esse aumento médio foi de 28,56 milissegundos.

Dado que os tempos obtidos mostram que a sobrecarga média, medida nos testes feitos na linguagem Java, foi um pouco menor que a obtida nos testes feitos com a linguagem Lua, uma pequena consideração é necessária. O que acontece é que na linguagem Java podemos cadastrar vários interceptadores, então a inclusão do mecanismo de transações apenas cadastra mais um interceptador no ORB. Todo o custo do mecanismo de interceptadores CORBA já está incluído no tempo de execução dos métodos, pois o *Openbus* já faz uso dos interceptadores CORBA.

Já em Lua, não é possível o uso de vários interceptadores, sendo necessário a utilização de um gerenciador de interceptadores para permitir a inclusão de outro interceptador CORBA. A sobrecarga devido ao mecanismo de transações é um pouco maior, pois inclui o código de gerenciamento de interceptadores. Assim, em Java é menor a diferença percebida após a inclusão do mecanismo de transações. No entanto, a diferença é de pouco mais de 3.5 milissegundos e não chega a ser significativa.

Método	**Tempo de resposta (ms)**	
	Instrumentado	**Não instrumentado**
register	118.98	88.86
removeMember	605.86	578.08
unregister	39.86	17.26
logout	26.58	1.29
getChallenge	27.76	1.86
loginByCertificate	50.68	19.02
findByCriteria	25.99	0.95
find	25.87	0.80
getIdentifier	16.52	1.05
createSession	1358.86	1337.18
getSession	18.57	1.02
getMembers	18.59	0.97
addObserver	21.13	3.80
addCredentialToObserver	30.20	1.18
removeCredentialFromObserver	28.85	1.21
renewLease	30.80	0.85
isValid	33.50	0.63
getSystemDeployment	19.02	1.00
areValid	38.08	4.81
_non_existent	28.53	0.45

Tabela 5.3: Tabela com os valores do tempo médio de resposta

Finalmente, devemos destacar que o impacto da sobrecarga é tanto maior quanto menor for o tempo médio de execução de um método remoto. Para métodos com grande tempo de processamento a sobrecarga será percentualmente menor e não será significativa. Afinal, se um dado método possui tempo médio de execução da ordem de alguns segundos, alguns milissegundos de sobrecarga terão impacto quase imperceptível.

Mas, no caso de invocações de funções com tempo menor de execução (da ordem de poucos milissegundos) e que são chamadas um grande número de vezes, em um laço de execução por exemplo, é necessário avaliar se o monitoramento do sistema trará benefícios que justifiquem a sobrecarga imposta.

Método	**Sobrecarga**	
	Diferença (ms)	**Percentual**
register	30.12	33.89%
removeMember	27.79	4.81%
unregister	22.60	130.93%
logout	25.29	1957.24%
getChallenge	25.90	1394.17%
loginByCertificate	31.66	166.48%
findByCriteria	25.04	2641.38%
find	25.07	3131.50%
getIdentifier	15.48	197.80%
createSession	21.67	1.62%
getSession	17.55	1719.20%
getMembers	17.63	1824.09%
addObserver	17.33	456.15%
addCredentialToObserver	29.02	2466.68%
removeCredentialFromObserver	27.65	2292.60%
renewLease	29.96	3533.88%
isValid	32.88	5256.65%
getSystemDeployment	18.02	1799.86%
areValid	33.27	691.26%
_non_existent	28.08	6203.97%

Tabela 5.4: Tabela com os valores médios da sobrecarga nos tempos de resposta

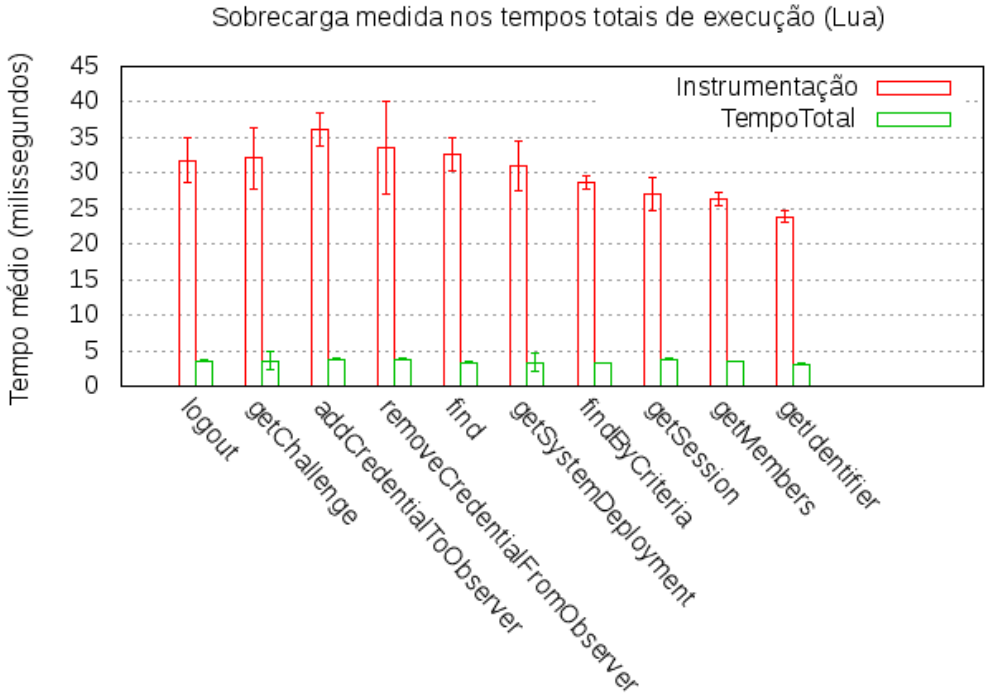

Figura 5.3: Comparação entre os tempos totais de execução (Testes feitos em Lua)

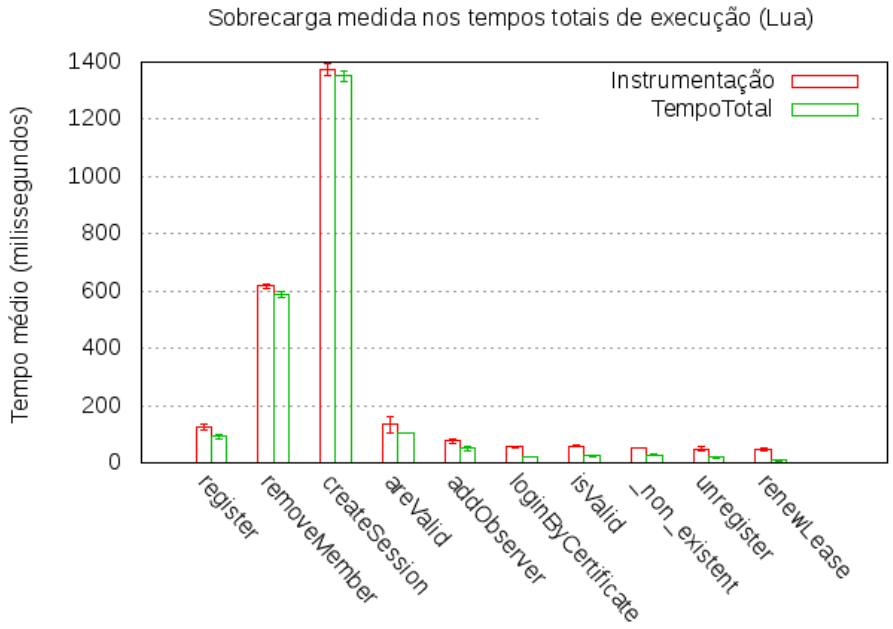

Figura 5.4: Comparação entre os tempos totais de execução dos métodos (Testes feitos em Lua)

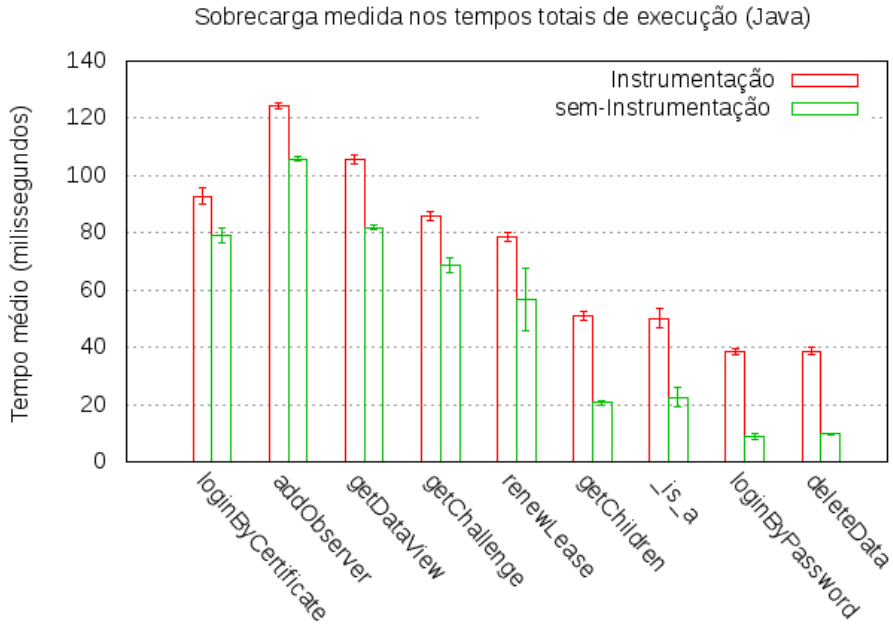

Figura 5.5: Comparação entre os tempos totais de execução dos métodos (Testes feitos em Java)

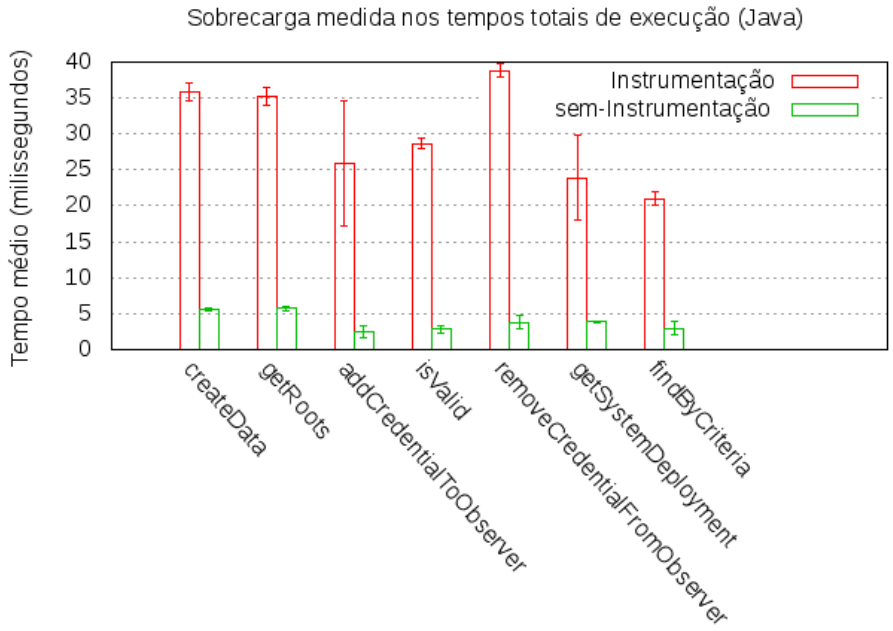

Figura 5.6: Comparação entre os tempos totais de execução dos métodos (Testes feitos em Java)

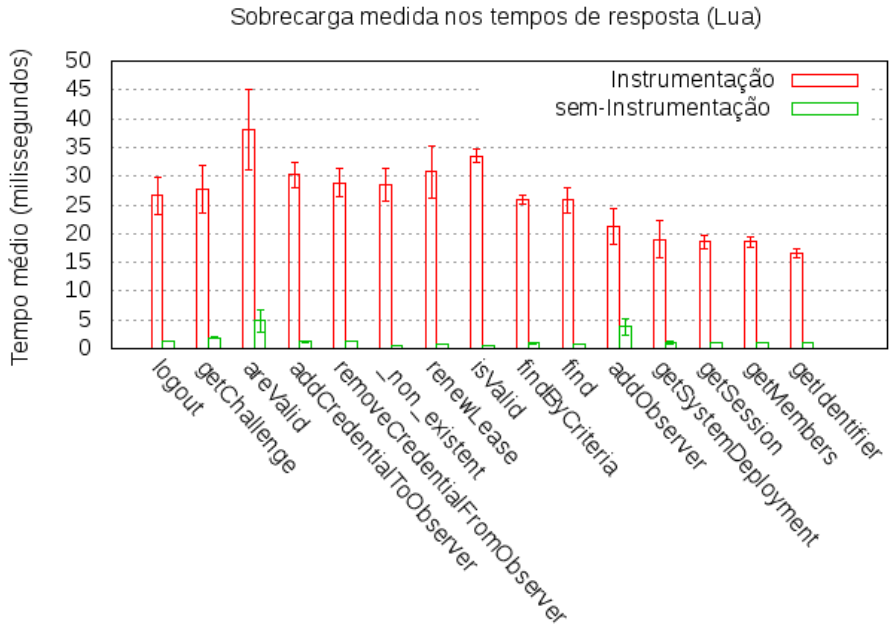

Figura 5.7: Comparação entre os tempos de resposta dos métodos(Testes feitos em Lua)

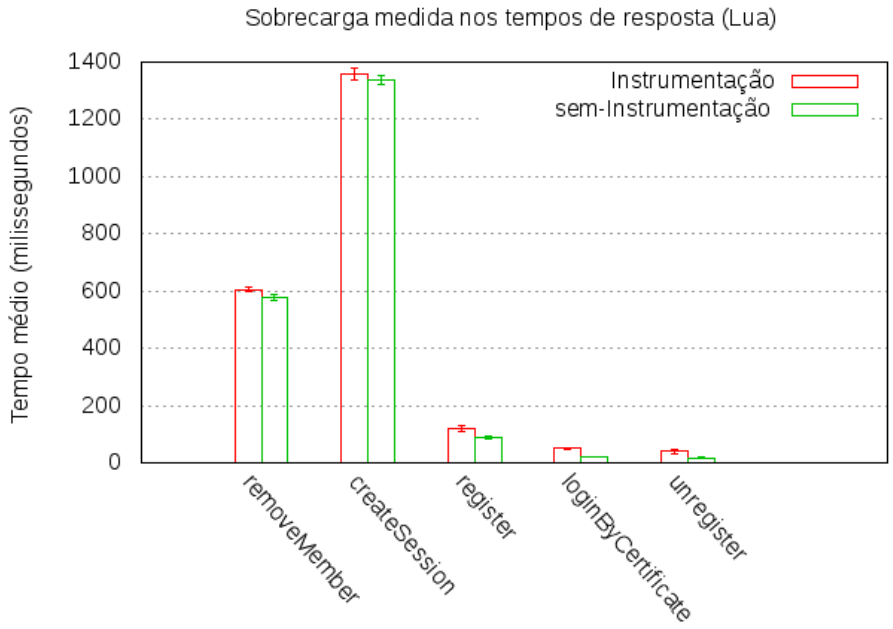

Figura 5.8: Comparação entre os tempos de resposta dos métodos (Testes feitos em Lua)

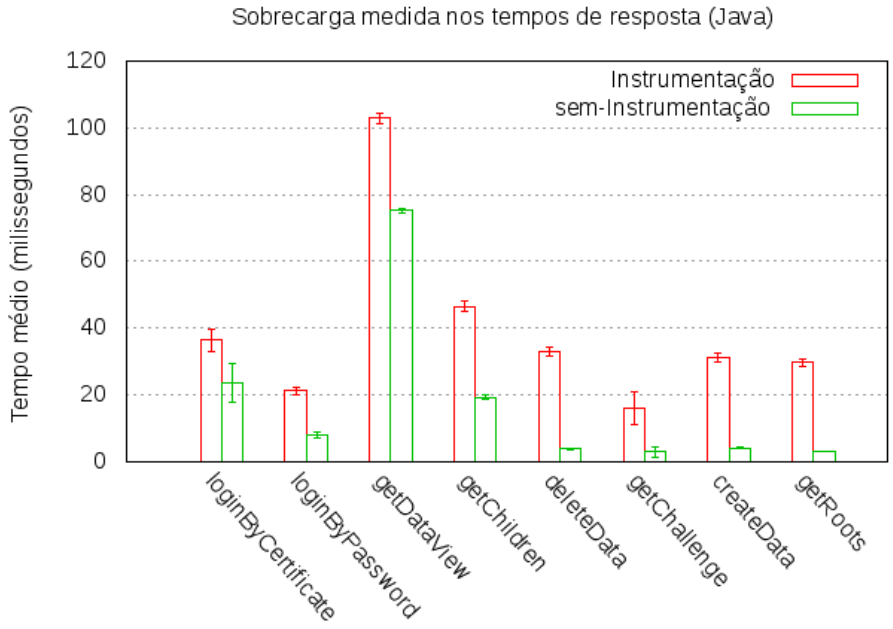

Figura 5.9: Comparação entre os tempos de resposta dos métodos (Testes feitos em Java)

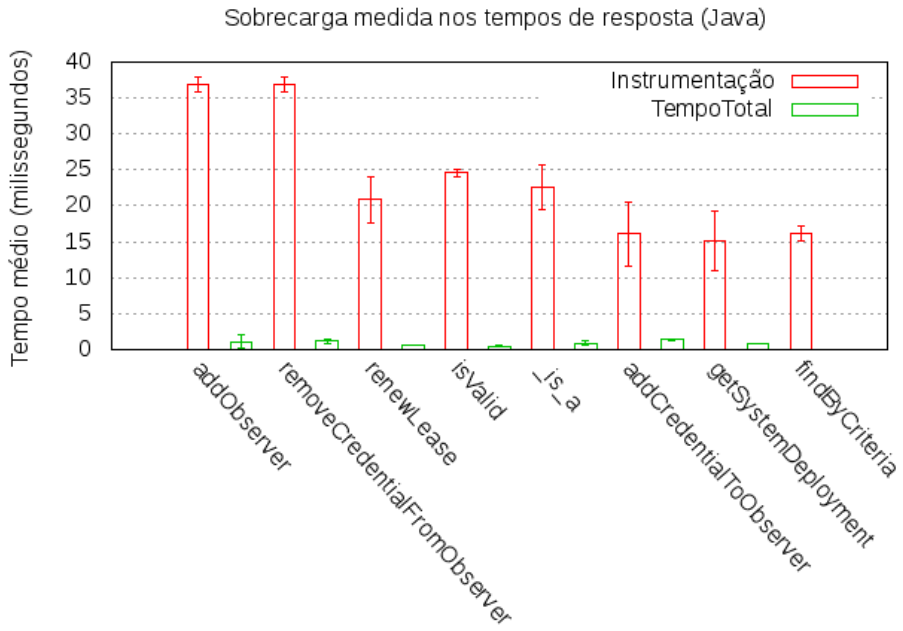

Figura 5.10: Comparação entre os tempos de resposta dos métodos (Testes feitos em Java)