Universidade do Minho
Escola de Engenharia

Luís Miguel Chaves Claro

Arquitectura Orientada a Eventos

Outubro de 2010

Universidade do Minho
Escola de Engenharia

Luís Miguel Chaves Claro

Arquitectura Orientada a Eventos

Dissertação de Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação do
Professor Doutor António Manuel Nestor Ribeiro
Engenheiro Jorge Miguel Barbosa
Patrão

Outubro de 2010

Agradecimentos

Ao longo da elaboração desta dissertação muitas foram as pessoas que de forma directa ou indirecta contribuíram para a mesma. A todas elas os meus sinceros agradecimentos.

Quero agradecer aos meus orientadores, Professor António Manuel Nestor Ribeiro da Universidade do Minho e ao Eng.º Jorge Miguel Barbosa Patrão na PT Inovação por todo o apoio prestados na elaboração da dissertação.

A todos os elementos do Departamento de Redes Inteligentes da PT Inovação pelo apoio prestado.

A todos os meus amigos e colegas pelo companheirismo e amizade.

À minha família pela compreensão, conselhos, incentivo e apoio incondicional prestado ao longo de toda a elaboração da dissertação.

Resumo

Actualmente, na área das TI, a preocupação na integração dos sistemas de negócio tem sido uma prioridade, permitindo que o processo de desenvolvimento de novas aplicações e serviços seja realizado de uma forma mais rápida e fácil. Tipicamente, as aplicações e produtos desenvolvidos são constituídos por múltiplos serviços, sendo portanto essencial criar sistemas de monitorização, que identifiquem situações anómalas e de rápida intervenção que suportem estes níveis de complexidade e heterogeneidade assegurando requisitos de tempo real.

Os operadores de telecomunicações enquadram-se neste cenário, caracterizados por fornecerem diferentes portfólios de serviços a um conjunto de clientes cada vez mais amplo. Em função desta constante evolução, há uma crescente necessidade de criar mecanismos de monitorização que se adaptem facilmente à dimensão dos serviços e clientes, isto é, soluções escaláveis que permitam continuar a garantir os níveis de serviço acordados.

Desta forma, requisitos como alta disponibilidade, fiabilidade, suporte transaccional e segurança tornam-se indispensáveis para o estudo de uma arquitectura de monitorização que suporte o processamento de múltiplos eventos, com informação acerca de diversos serviços com diferentes critérios de actuação.

A presente dissertação, tem como principal objectivo a investigação e concepção de uma arquitectura orientada a eventos que permita uma eficiente implementação de módulos de monitorização em sistemas de telecomunicações no contexto do sistema NGIN Manager, responsável pela gestão de subsistemas e aplicações da plataforma NGIN desenvolvida pela PT Inovação. A nova arquitectura resolve problemas existentes na solução actual como a falta de escalabilidade e fiabilidade na comunicação com os componentes da rede inteligente (IN) e introduz mecanismos que facilitam a gestão e controlo de recursos aplicacionais e a implementação de novos módulos de monitorização.

Abstract

Actually, in the IT area, the concern in business system integration has been a priority, allowing the development process of new applications to be performed more quickly and easily. Typically, developed products and applications are consisted of multiple services, therefore, it is essential to create monitoring systems that identify abnormal situations and intervene quickly to support these levels of complexity and heterogeneity, ensuring real-time requirements.

Telecoms operators fall into this scenario, characterized by providing different service portfolios to an increasingly broad set of customers. Due to this constant evolution, there is a growing need to establish monitoring mechanisms that adapt easily to services and clients size, i.e., scalable solutions that can continue to guarantee the agreed service levels.

Thus, requirements like high availability, reliability, transaction support and security become indispensable to the study of a monitoring architecture that supports the processing of multiple events, with information about various services with different performance criteria.

This dissertation has as main objective the research and development of an event-driven architecture that enables an efficient implementation of monitoring modules in telecommunication systems within the NGIN Manager system, responsible for the management of subsystems and applications of the NGIN platform developed by PT Inovação. The new architecture solves existing problems in the current solution as the lack of scalability and reliability in communication with intelligent network (IN) components and introduces mechanisms that make easier the management and control of applicational resources and the implementation of new monitoring modules.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AMQP	Advanced Messaging Queuing Protocol
ASN.1	Abstract Syntax Notation One
ASM	Automatic Storage Management
BER	Basic Encoding Rules
BPM	Business Process Management
CEP	Complex Event Processing
CER	Canonical Encoding Rules
CORBA	Common Object Request Broker Architecture
DER	Distinguished Encoding Rules
DSCF	Data Session Control Function
DSCP	Data Session Control Point
EAI	Enterprise Application Integration
EJB	Enterprise Java Bean
ESB	Enterprise Service Bus
EDA	Event Driven Architecture
GSER	Generic String Encoding Rules
HP-UX	Hewlett Packard UniX
IN	Intelligent Network
INAP	Intelligent Network Application Part protocol

IVR	Interactive Voice Response
JMS	Java Message Service
JTA	Java Transaction API
JTS	Java Transaction Service
MIB	Management Information Base
MDB	Message Driven Bean
MOM	Message Oriented Middleware
NGIN	Next Generation Intelligent Network
OMG	Object Management Group
ORB	Object Request Broker
OTS	Object Transaction Service
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
OCI	Oracale Call Interface
PER	Packed Encoding Rules
SCF	Service Control Function
SCL	Service Control Logic
SCP	Service Control Point
SDF	Service Data Function
SDP	Service Data Point
SEDA	Staged Event-Driven Architecture
SMP	Service Management Point
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
STOMP	Streaming Text Oriented Messaging Protocol

SRF	Specialized Resource Function
UDDI	Universal Description Discovery and Integration
VSSP	Virtual Service Switching Point
WSRF	Web Services Resource Framework
WS-*	Web Services specifications
XDR	eXternal Data Representation
XER	XML Encoding Rules
XML	eXtensible Markup Language

Capítulo 1

Introdução

O ramo das telecomunicações é um sector extremamente competitivo, onde a diferença é marcada cada vez mais, não só, pelo serviço em si, como também, pela qualidade do serviço. As redes inteligentes enquadram-se neste cenário, caracterizadas por possuírem serviços cuja operacionalidade deve ser constante e onde todas as falhas sejam reportadas para plataformas de gestão.

A solução NGIN [1], desenvolvida pela PT Inovação, é uma plataforma que permite aos operadores telefónicos disponibilizar serviços de voz ou dados, pré-pagos ou pós-pagos, aos seus clientes fixos ou móveis, em redes de comutação de circuitos, comutação de pacotes ou convergentes e onde o produto NGIN Manager possui um papel fundamental como plataforma de gestão.

À data, a solução NGIN suporta os serviços de rede inteligente (Pré-Pagos, Pós-Pagos com Controlo de Custos, VPN, Número Verde, Serviços de Valor Acrescentado, Acesso Internet Móvel 3G, Portais de Conteúdos, etc.) de operadores como PT Comunicações, SA. (Portugal), TMN, Telecomunicações Móveis, SA. (Portugal), UZO (Portugal), Phone-ix (Portugal), VIVO (Brasil), UNITEL (Angola), Meditelecom (Marrocos), CVT, Cabo Verde Telecom (Cabo Verde), CST, Companhia Santomense de Telecomunicações (São Tomé e Príncipe), Mascom Wireless (Botswana) e Timor Telecom (Timor Leste).

Os serviços apresentam uma percentagem reduzida de dependência da rede, normalmente restrita à Lógica de Controlo de Sinalização (SCL). Os aspectos de negócio são tratados directamente pelas lógicas de serviço e, assim, mantidos à parte dos módulos que suportam a interacção com a rede. Da perspectiva de rede, estes produtos abrangem duas áreas principais de controlo e

sinalização de serviços:

- as redes de comutação de circuitos, onde as capacidades requeridas são disponibilizadas nas solução de comutação de serviços (NGIN vSSP) e controlo de serviços (NGIN SCP);
- as redes de comutação de pacotes.

O subsistema NGIN vSSP possibilita a coexistência com outras soluções em redes inteligentes (IN), de forma a que novos serviços possam ser explorados sem necessidade de actualização para as IN ao nível da rede. Quando integrada com o subsistema NGIN SCP permite a disponibilização de um conjunto completo de serviços, podendo ser integrado com plataformas de controlo (SCP) de outros vendedores. Com o NGIN SCP são suportadas todas as entidades funcionais específicas para redes inteligentes, isto é, as funcionalidades de controlo de serviços (SCF) para a execução da lógica do serviço e funcionalidades de recursos específicos (SRF) para o controlo de resposta interactiva de voz (IVR). O subsistema NGIN SDP é responsável por armazenar, em base de dados, informação necessária para a execução da lógica do serviço IN. A informação armazenada em base de dados contém tipicamente dados dos clientes, configurações de serviços e/ou de rede. O acesso às bases de dados é da responsabilidade das funcionalidades dos serviços de dados (SDF), permitindo o seu uso pela lógica de controlo de sinalização (SCL) efectuada nos SCFs e DSCFs.

Na Figura 1.1 é apresentada a execução da lógica de serviços, em redes de voz e dados. A interacção com a rede é limitada à lógica de controlo de sinalização (SCL), permitindo que a lógica de serviço fique sob responsabilidade dos componentes de controlo (SCF e DSCF). Estes, por sua vez, utilizam as funcionalidades dos serviços de dados do subsistema SDP para obter informação existente em base de dados.

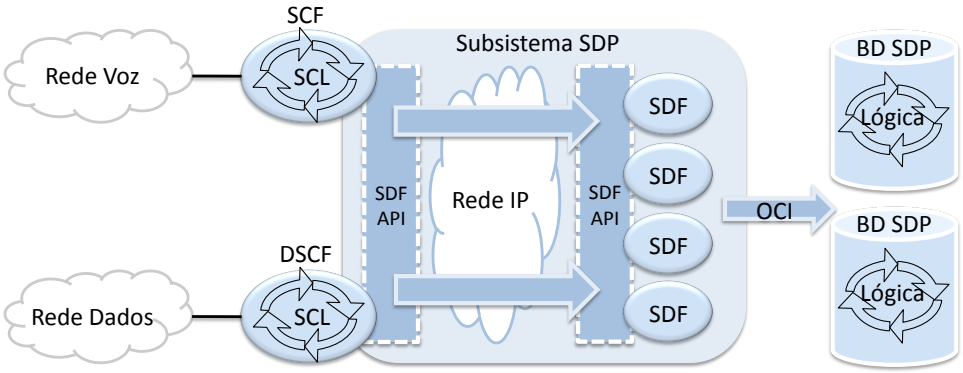

Figura 1.1: Execução da lógica de serviço

A solução NGIN Manager é uma plataforma de gestão de aplicações e subsistemas NGIN (p.e.: SCP, SDP, SMP, VSSP, DSCP, etc.). Esta plataforma é constituída por 4 subsistemas de gestão: configuração, desempenho, falhas e monitorização em tempo real. Em cada um dos subsistemas, a interface com os componentes da solução NGIN é feita por intermédio de um conjunto de agentes, responsáveis pela:

- Sincronização automática de inventário (informação de catalogação dos componentes suportados);
- Detecção de falhas e geração de alarmes;
- Recolha de contadores de desempenho de longo termo;
- Recolha de contadores de desempenho de tempo real.

O módulo de gestão de configuração é denominado de **NGIN Config** e é responsável pela monitorização e configuração da informação de registo dos componentes suportados pela solução NGIN. Baseado num meta modelo definido em XML, o módulo NGIN Config modela todas as entidades NGIN hierarquicamente organizadas, gerando automática e periodicamente informação sobre os componentes existentes em todos os nós NGIN. A hierarquia encontra-se dividida em topologia física, entidades lógicas, serviços e processos de negócio.

A gestão de desempenho recorre ao módulo **NGIN Perf** e a um conjunto de aplicações responsáveis pela recolha de contadores de desempenho. Os contadores de desempenho são aplicações distribuídas, existindo um para cada tipo de aplicação gerida ou funcionalidade NGIN, que valida e envia os contadores recolhidos para o módulo NGIN Perf. Este módulo, por sua vez, com base em limites definidos, detecta quando estes não são respeitados e gera alarmes que são enviados para o módulo **NGIN Fault**.

A gestão de alarmes engloba a parte infra-estrutural e aplicacional. A gestão de alarmes infra-estrutural monitoriza hardware, rede, software de base do qual o NGIN depende (sistema operativo, base de dados e servidores aplicacionais) e gera alarmes, por exemplo, para indicadores de carga de CPU, taxa de ocupação da memória, taxa de ocupação do sistema de ficheiros associados a uma máquina, ocupação das tablespaces, taxa de ocupação da gestão automática de armazenamento nas base de dados (ASM), congestão das filas de espera lógicas e relatórios de erros nas base de dados. A gestão de alarmes aplicacional baseia-se na taxa de sucesso/insucesso da execução de operações, por exemplo, para chamadas de voz tratadas num SCF e sessões de

dados num DSCF pode ser calculado o número de invocações e terminações, estado das chamadas/sessões, número de chamadas/sessões rejeitadas, número de chamadas/sessões em simultâneo e por segundo, tempos de execução SQL, entre outros.

Os alarmes podem ter acções imediatas de acordo com regras específicas ou por transgressão de limites configurados. A informação estatística recolhida é processada e armazenada numa data warehouse especializada.

A gestão de falhas recorre a **agentes NGIN** para diagnóstico de falhas e geração de alarmes e ao módulo **NGIN Fault** para a recepção, processamento, armazenamento e visualização dos alarmes. Os agentes NGIN detectam falhas específicas de cada componente funcional e recolhem contadores indicativos do seu desempenho. O módulo NGIN Fault permite executar acções automáticas ou manuais sobre os alarmes, como por exemplo o envio de correio electrónico ou de uma mensagem de texto em função de determinado alarme.

A monitorização em tempo real recorre ao módulo **NGIN MTR** e consiste na disponibilização de interfaces cliente específicas que permitem a monitorização em tempo real de um conjunto de estatísticas de curto prazo. Estas estatísticas caracterizem o estado de funcionamento interno de um dado componente NGIN, permitindo concluir sobre o desempenho e dimensionamento do componente observado.

Os **Agentes e Colectores NGIN** disponibilizam interfaces para os módulos NGIN Fault, NGIN Perf, NGIN Config e NGIN MTR em sistemas operativos LINUX e HP-UX. Os Agentes NGIN modelam cada componente funcional da solução, detectando falhas específicas de cada um e recolhendo contadores indicativos do seu desempenho. Os Colectores NGIN são específicos para cada tipo de aplicação gerida ou funcionalidade NGIN e validam e enviam os contadores de desempenho recolhidos para o módulo NGIN Perf.

Inserido num mercado extremamente concorrente e em constante evolução, não é tolerável que ocorram falhas em componentes ou serviços NGIN sem haver subsistemas capazes de detectar estas situações. Neste contexto, as plataformas de gestão e monitorização têm que acompanhar esta evolução, tornando-se necessário o estudo de uma arquitectura eficiente e facilmente integrável com novos módulos.

1.1 Motivação e objectivos

Como se referiu atrás, os operadores de telecomunicações enquadram-se num vasto e potencial mercado, onde a concorrência é baseada cada vez mais no desenvolvimento de novos portfólios de serviços ao invés da concorrência baseada nos preços. Neste ambiente de mercado extremamente competitivo a flexibilidade e fiabilidade da solução NGIN é crucial para o sucesso do negócio. Muitos destes serviços, são vendidos a inúmeros clientes, cada um com configurações diferentes, que esperam uma disponibilidade permanente - por exemplo a TMN em 29 de Setembro de 2009 tinha mais de 7 milhões de clientes. Com a monitorização de serviços torna-se possível detectar problemas físicos/lógicos, criar relatórios do estado dos serviços e, quando possível, prevenir possíveis falhas com previsão de comportamento. Neste sentido, com a facilidade de integração de novos serviços é essencial dotar as plataformas de monitorização com mecanismos que facilitem o suporte dos novos serviços e com garantias de requisitos como fiabilidade, disponibilidade, segurança e suporte transaccional. A solução NGIN Manager passou por estratégias diferentes no seu processo de evolução, no entanto continuam-se a verificar situações em que é difícil garantir os requisitos necessários a um cenário livre de falhas. O principal problema está na forma como os colectores NGIN enviam a informação de monitorização das máquinas em produção nos clientes para os módulos responsáveis por analisar a informação, responsáveis por criar relatórios de desempenho de tempo real e longo termo, detecção de falhas e geração de alarmes, etc.

Uma das estratégias consiste na transferência de ficheiros, isto é, são utilizados ficheiros na máquina monitorizada para armazenar a informação de monitorização recolhida, sendo posteriormente enviados para o NGIN Manager para proceder à análise dos resultados. Esta estratégia está sujeita às seguintes desvantagens:

- Corrupção dos ficheiros de output;
- Falha nas transferências de ficheiros;
- Alterações nos relatórios obrigam a evolução dos parsers;
- Implicam a transferência de volumes elevados de informação.

Uma alternativa a esta solução consiste na utilização do protocolo SNMP, sendo que desta forma a informação de monitorização é escrita numa MIB própria, permitindo posteriormente ao

colector obter a informação necessária. Esta estratégia, contudo, não resolve todos os problemas e apresenta vantagens e desvantagens que se explicam de seguida:

- Vantagens:
  - Método padronizado de acesso à informação;
  - Simplificação da descoberta de inventário.
- Desvantagens:
  - Não foi pensado para ser 100% fiável:
    - Aplicações podem falhar na escrita da informação de monitorização na MIB;
    - Colecta pode falhar na leitura da MIB;
    - Pouca performance na estratégia de persistência da informação, podendo ocorrer perdas de informação por parte dos agentes que escrevem a informação na MIB.
  - Problemas na instalação nos clientes:
    - Obriga à instalação de NetSNMP e patches S.O. associados;
    - Em certas máquinas (HP-UX) ocorrem problemas na instalação/compilação do NetSNMP.

Para estes problemas existem algumas abordagens que poderiam melhorar a estratégia como dotar o agente de escrita na MIB com capacidade de persistir a informação resolvendo as falhas de leitura e rede, utilizando escritas repetidas entre tentativas sem resultado para resolver falhas na escrita e através do uso de aplicações SNMP desenvolvidas/compiladas em máquinas diferentes para resolver os problemas de instalação. No entanto, o esforço elevado na melhoria da estratégia SNMP não daria garantias de resolução de todos os problemas.

Tendo em conta os problemas das estratégias anteriores, a complexidade das configurações das plataformas associadas, necessidades de performance e notificações imediatas com inventários específicos, torna-se indispensável ter em conta novas iniciativas, abordagens tecnológicas e modelos relevantes para o desenvolvimento de uma arquitectura orientada a eventos (EDA) que permita a rápida integração com novos serviços, com garantias de fiabilidade na transferência de informação de monitorização e que permita a rápida integração de novas ferramentas de monitorização no NGIN Manager. A capacidade de integração da arquitectura deverá suportar também diferenças ao nível das linguagens de programação, isto é, os serviços a monitorizar poderão ser desenvolvidos com linguagens diferentes da nova arquitectura.

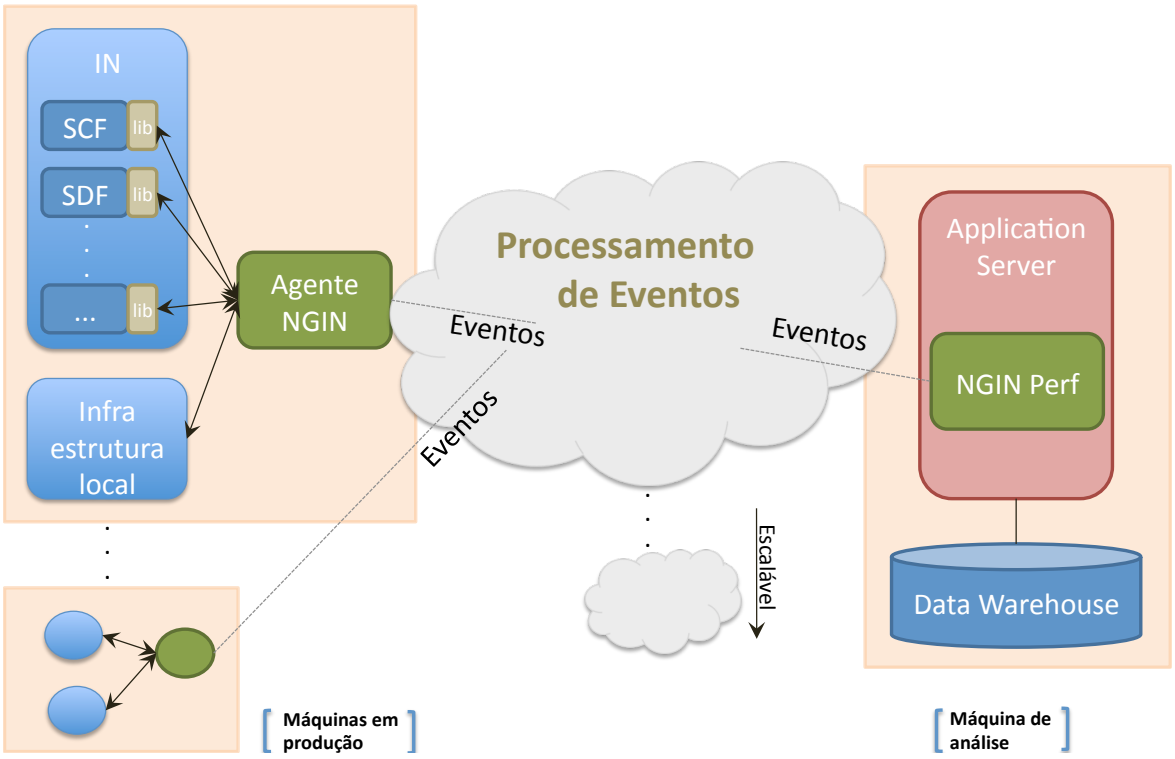

Figura 1.2: Integração prevista do NGIN Manager com arquitectura EDA

Uma EDA é uma boa estratégia pois, permite o envio de informação de forma assíncrona ao contrário de sistemas que têm que estar constantemente a interrogar o sistema por alterações. Com esta arquitectura não é afectado o funcionamento lógico dos vários módulos existentes, enquadrando-se perfeitamente numa arquitectura orientada a serviços (SOA). A EDA introduz um meio seguro, eficaz e fiável de comunicação entre os vários módulos, tratando a troca de informação como eventos e introduzindo módulos capazes de os processar.

Dado que a arquitectura não deve sobrecarregar as máquinas em produção nos clientes a persistência da informação deverá ser realizada num nó intermédio entre os agentes NGIN e máquina de recolha de informação, tirando proveito da existência de topologias com várias máquinas em produção nos clientes, tornando o sistema mais escalável, conforme apresentado na Figura 1.2.

Desta forma, o estudo da arquitectura orientada a eventos para monitorização e supervisão da plataforma NGIN, tem como principais objectivos:

- Avaliar o "estado da arte" da arquitectura NGIN e do subsistema NGIN Manager;

- Levantar os requisitos necessários à especificação de uma nova arquitectura;
- Levantar o "estado da arte" das arquitecturas Event-Driven mais adequadas aos requisitos;
- Avaliar as plataformas que melhor se enquadram na arquitectura;
- Avaliar o padrão de comunicação da arquitectura Event-Driven.
- Contribuir com um formato e estrutura das mensagens na comunicação entre os diversos componentes que interagem com a arquitectura;
- Contribuir com uma arquitectura Event-Driven que responda eficazmente às necessidades da solução NGIN Manager, extremamente importantes dada a dimensão das operadoras de telecomunicações, permitindo a rápida integração com novos serviços, garantindo requisitos como fiabilidade, disponibilidade, segurança e permitindo um rápido desenvolvimento de ferramentas de monitorização.

1.2 Organização da dissertação

Esta dissertação está organizada em cinco capítulos, Introdução, Estado da Arte, Concepção e implementação da solução, Cenários de utilização e Testes e Conclusões.

No presente capítulo é apresentada a área em que a dissertação se insere, descrevendo os serviços da solução NGIN que o subsistema NGIN Manager monitoriza e supervisiona. Os componentes e funcionalidades da solução NGIN Manager são explicados, apresentando-se posteriormente os problemas existentes com abordagens em curso e o porquê da necessidade de uma arquitectura orientada a eventos.

No capítulo 2 é apresentado o estado da arte das arquitecturas Event-Driven, focando as necessidades existentes no subsistema NGIN Manager. Neste sentido, são abordados as principais alternativas de plataformas para comunicação por eventos, focando as funcionalidades dos intermediários de mensagens e as técnicas de processamento de eventos. Para as arquitecturas Event-Driven, tendo em conta as necessidades de integração do NGIN Manager são apresentadas estratégias, protocolos de comunicação e padrões existentes que visam a interoperabilidade entre plataformas distintas.

No capítulo 3 é inicialmente feita uma contextualização com a solução actual do NGIN Manager na PT Inovação, posteriormente é apresentada a análise de requisitos, a arquitectura da

nova solução, a especificação da estrutura das mensagens e os passos necessários para o desenvolvimento dos componentes de envio, intermediação e análise de eventos.

No capítulo 4 são apresentados alguns cenários de utilização da solução desenvolvida e os resultados de testes efectuados que demonstram a validade da solução tendo em conta os requisitos.

No capítulo 5 são descritas as conclusões resultantes do trabalho realizado e são avaliadas as contribuições ao subsistema NGIN Manager conseguidas com o desenvolvimento da solução e testes realizados. Os possíveis trabalhos futuros são também tidos em conta, de forma a integrar novas funcionalidades na solução.

Capítulo 2

Estado da Arte

Neste capítulo é apresentado o estado da arte das arquitecturas Event-Driven, sendo descritas as suas vantagens e modelos de interacção entre intermediários, publicadores e subscritores. As plataformas de mensagens existentes são apresentadas, assim como, frameworks que permitam a implementação de serviços nessas plataformas, comparação geral de algumas destas plataformas e modelos de processamento de eventos. Para além dos modelos de uma EDA as necessidades do NGIN Manager requerem também modelos de integração, pelo que são descritas as abordagens existentes no âmbito de padrões EAI e plataformas integradas de mensagens que se adaptem às arquitecturas Event-Driven.

2.1 Arquitectura Orientada a Eventos

Uma arquitectura orientada a eventos (EDA) caracteriza-se por uma solução que permite implementar sistemas, serviços e aplicações cuja comunicação é baseada na reacção a eventos. Estas arquitecturas são caracterizadas por possuírem módulos de produção e consumo de eventos, onde cada módulo não tem necessariamente de saber da existência do outro. Os eventos são encaminhados através de canais próprios, quando os consumidores não estão disponíveis os eventos são persistidos e assim que fiquem disponíveis os eventos são entregues aos consumidores. Este mecanismo é denominado de "store and forward" [2].

As vantagens de uma EDA estão assentes no conceito de diferentes n-para-n interacções desassociadas entre publicadores e subscritores de eventos [2], tal como:

- Desassociação de espaço: publicadores e subscritores não têm de saber da existência de cada um.

- Desassociação de tempo: não é necessário que os subscritores de eventos estejam sempre activos à espera de receber os eventos de publicadores, o canal de eventos armazena e reencaminha-os quando os subscritores estiverem disponíveis;
- Desassociação de sincronização: publicadores e subscritores não precisam de estar sincronizados para gerar e receber eventos. A lógica de negócio ocorre concorrentemente com o encaminhamento dos eventos.

Além disso, a comunicação por eventos leva a uma implementação modular de serviços e componentes, beneficiando da sua reutilização em contextos diferentes.

2.1.1 Interacções com o canal de eventos

O canal de eventos é responsável por encaminhar eventos dos produtores para os consumidores de eventos. Contudo, existem duas estratégias de implementação diferentes, "queues" e "topics". Um factor importante do canal de eventos, em sistemas com inúmeros eventos e com tipos heterogéneos é a capacidade de filtrar os eventos pretendidos de acordo com os requisitos dos consumidores de eventos.

Queues de Eventos

Este modelo fornece um encaminhamento ordenado dos eventos, onde cada evento é removido da queue e enviado a apenas um consumidor. O produtor de eventos não necessita que no momento de envio de um evento estejam consumidores activos. Neste caso, os eventos são persistidos e posteriormente encaminhados ao primeiro que fique activo. Desta forma, as queues podem ser vistas como uma forma de balancear a carga de processamento dos eventos por vários consumidores, dado que cada um recebe eventos distintos.

Na Figura 2.1 é apresentada a estrutura básica de uma queue com capacidade para persistir eventos, o produtor, o espaço lógico onde os eventos são persistidos e o consumidor.

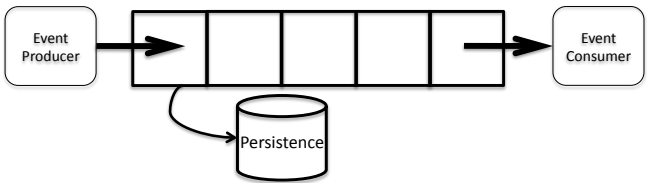

Figura 2.1: Queue de Eventos

Tópicos de Eventos

Um tópico é um repositório de eventos que se baseia no conceito de "publish/subscribe". A associação entre os publicadores e os subscritores de cada tópico é de muitos para muitos, para os quais existem diferentes métodos que podem ser usados para filtrar eventos [2].

A ideia original consiste no uso de temas e grupos, permitindo associar uma descrição ao canal de eventos que os publicadores e subscritores podem usar para comunicar. Desta forma as operações fundamentais são publish() para disseminar um evento para um grupo específico do canal, subscribe() para ficar associado a um grupo e unsubscribe() para terminar a associação com um grupo. Uma estratégia derivada consiste na utilização de uma hierarquia de grupos, onde uma subscrição num nó superior inclui todos os sub nós.

Outro método que pode ser usado ou em complemento de outros métodos para distinguir eventos consiste em filtrar o conteúdo dos eventos, isto é, no momento da subscrição identificam-se os eventos pretendidos. De forma a identificar os eventos podem-se usar palavras-chave que correspondam a determinadas propriedades, objectos modelo que incluam mais propriedades ou através de código executável enviado no objecto de uma interface específica que permita ser invocado no momento de filtrar os eventos.

Uma técnica diferente que pode ser usada baseia-se na existência de tipos de eventos, tirando partido dos tipos das linguagens de programação orientadas a eventos.

Uma variação deste modelo "publish/subscribe" permite que o consumidor invoque métodos remotos no produtor, o produtor gera o evento e retorna-o ao consumidor.

Na Figura 2.2 são apresentados os elementos básicos de um tópico de eventos, o publicador de eventos, o filtro e o subscritor de eventos.

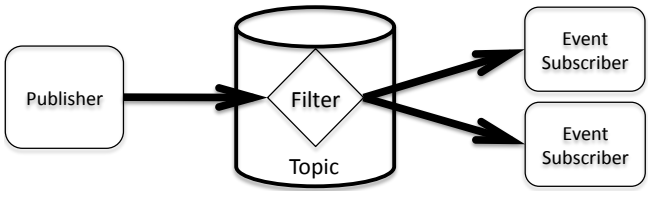

Figura 2.2: Tópicos de Eventos

2.1.2 Plataformas de Mensagens

Devido à importância que pode estar associada a um evento, a submissão e entrega de eventos deve ser realizada garantindo requisitos fundamentais. Por exemplo, em sistemas de monitorização, um evento pode estar associado a um alarme crítico, tornando-se fundamental que os eventos sejam fidedignos e que não ocorram falhas no seu envio. Desta forma, existem diferentes alternativas de serviços de mensagens com diferentes funcionalidades, das quais se destacam os Web Services (WS) [3], Open Grid Services Infrastructure (OGSI) [4], Object Request Broker (ORB) [5, 6] e as Message Oriented Middleware (MOM). Os protocolos Advanced Message Queuing Protocol [7], Java Message Service [8], Streaming Text Oriented Message Protocol [9] e OpenWire [10] são exemplos de MOMs, "middleware" usado para a troca de mensagens. Cada um tem diferentes funcionalidades, contudo no âmbito de sistemas de monitorização as principais são o desempenho, fiabilidade, segurança e métodos que suportem o controlo da comunicação, como o uso de transacções [11]. O desempenho é um factor fundamental, dado que o intermediário de mensagens deve estar preparado para suportar a execução de operações com o mínimo de interrupções, garantindo todas as funcionalidades mesmo em momentos de maior carga. Dado que se trata de um elemento fundamental da arquitectura, o ideal seria que este fosse um componente facilmente escalável. A comunicação de forma assíncrona é também fundamental para obter melhores desempenhos.

A fiabilidade é essencial em comunicações assíncronas e de forma a garantir tolerância a falhas. Em sistemas de mensagens a fiabilidade pode ser garantida através de subscrições duráveis e através da persistência de eventos num intermediário [8]. Desta forma, mesmo que ocorram problemas no canal de eventos, estes encontram-se persistidos podendo ser reencaminhados logo que as falhas estejam corrigidas. As subscrições duráveis permitem que os eventos sejam sempre entregues aos subscritores, independentemente de no momento do envio do evento se encontrarem inactivos.

A segurança nos sistemas de mensagens deve assegurar quer a validade dos diversos componentes, publicador, intermediário e subscritor, quer a integridade e privacidade da informação dos eventos.

As transacções são úteis quer nos produtores, quer nos subscritores [2], permitindo que a publicação de um determinado grupo de eventos que só faça sentido em conjunto. Desta forma, é garantido aos publicadores que todos os eventos enviados serão recebidos em conjunto. Os subscritores usam transacções para receber um grupo de eventos assegurando que a transacção só termina quando todos os eventos forem processados. No caso da ocorrência de algum pro-

blema no envio de um ou mais eventos é realizado um "rollback" e todo o conjunto de eventos é retransmitido, caso contrário é realizado um "commit".

Nas seguintes secções são descritas as principais funcionalidades de cada sistema de mensagens.

Web Services

Os Web Services são independentes da plataforma, linguagem de programação e tipo de transporte. Existem dois padrões WS-* principais para arquitecturas EDA, WS-Eventing e WS-Notification [11]. As mensagens são baseadas em XML e encapsuladas em SOAP.

WS-Notifications tem três componentes principais, WS-BaseNotification que define a interacção entre publicadores e subscritores, WS-BrokeredNotification que define as interfaces de rede dos intermediários e WS-Topics que define um tópico hierárquico [3].

WS-Eventing é análogo ao WS-BaseNotification, no entanto não tem controlo nos intermediários e na hierarquia de tópicos.

A fiabilidade, segurança e suporte transaccional podem ser asseguradas através dos módulos WS-Reliability, WS-Security e WS-Transaction. O facto dos módulos WS-* usarem SOAP que é baseado em XML, pode afectar o desempenho através do tempo de execução da análise do conteúdo XML [12].

Estes padrões WS-* são suportados em várias frameworks, como Apache Muse, Apache Axis2 ou IBM WebSphere Application Server.

Open Grid Services Infrastructure

Com Open Grid Services Infrastructure (OGSI) [11] cria-se uma infra-estrutura OGSA, através da coordenação de serviços de uma grid computacional por meio de uma extensão a Web Services. OGSI permite criar, identificar e gerir instâncias de serviços, controlar os dados do estado do serviço, notificar alterações do estado do serviço, gerir serviços de colecta e tratar falhas.

OGSI define um serviço de notificações baseado no paradigma de publicação/subscrição, a subscrição das notificações é da responsabilidade do componente NotificationSink e as publicações do componente NotificationSource. OGSI é uma extensão dos Web Services e como tal, com a introdução da WSRF e do módulo WS-Notification, o OGSI foi substituído pela WSRF e o serviço de notificações da OGSI foi substituído pelo WS-Notification [4].

Object Request Broker

Um agente de requisição de objectos (ORB) permite a comunicação entre sistemas distribuídos de objectos através de um intermediário que procede à serialização dos objectos. Um dos ORBs mais populares é o CORBA. CORBA [11] foi criado pela OMG para simplificar as comunicações em sistemas distribuídos entre diferentes linguagens de programação e plataformas. Em comunicações de eventos distribuídas o CORBA tem como especificações os Serviços de Eventos (Event Service) e os Serviços de Notificação (Notification Service).

O Serviço de Eventos [5] define um canal de eventos e dois modelos diferentes de comunicação, "push" e "pull". O modelo "push" é caracterizado por ter publicadores de eventos que enviam assincronamente eventos a consumidores. No modelo "pull" o consumidor de eventos faz pedidos a um publicador de eventos.

O Serviço de Notificação amplia as funcionalidades do Serviço de Eventos permitindo filtrar eventos e introduz qualidade de serviço ao sistema de mensagens [6, 11].

A fiabilidade do CORBA pode ser assegurada através da configuração da fiabilidade dos eventos, da comunicação e do modo de entrega. A fiabilidade dos eventos e da comunicação pode ser definida como persistente, melhor esforço e com/sem confirmações para o modo de entrega. As transacções são suportadas pelo CORBA através do serviço de transacção de objectos (OTS) nas comunicações entre os publicadores e o canal de eventos e entre o canal de eventos e os subscritores. Em termos de segurança a OMG disponibiliza um conjunto de especificações de segurança que podem ser usadas. Para além de melhorias na qualidade de serviço, no serviço de notificações CORBA o desempenho e escalabilidade foram também aperfeiçoadas.

Java Message Service

O serviço de mensagens Java (JMS) [8, 11] disponibiliza uma API Java para criar, enviar, receber e ler mensagens de sistemas de middleware orientados a mensagens (MOM). As mensagens têm um tipo predefinido, podendo ser uma TextMessage, BytesMessage, StreamMessage, ObjectMessage ou MapMessage.

A API JMS pode ser implementada por outro sistema de mensagens para realizar as operações associadas ao transporte de eventos, por exemplo, as especificações WS-*.

Para que o JMS garanta a fiabilidade na troca de eventos os publicadores de eventos devem definir o modo de entrega como sendo do tipo persistente e as subscrições devem ser defini-

das como duráveis. As sessões JMS podem usar transacções de forma a garantir que um grupo de mensagens é recebido e processado em conjunto. Adicionalmente pode ser usada a API de transacções Java (JTA) que permite delimitar transacções distribuídas e o serviço de transacções Java (JTS) para permitir o uso de outros serviços de transacções. A privacidade e integridade das mensagens são deixadas à responsabilidade das implementações disponibilizadas pelos fornecedores, no entanto, o JMS oferece funcionalidades básicas como autenticação nas conexões. A performance do JMS depende das implementações e do uso específico a que se destina, isto é, dos tipos de evento, tamanho da informação contida no evento e do tipo de filtro usado no canal de eventos.

A API JMS é usada pela plataforma Java EE através de Message Driven Beans (MDB) [13], estes permitem a comunicação assíncrona entre Enterprise Java Beans (EJB). Existe também um conjunto de plataformas Java, como o Apache ActiveMQ, que permitem implementar sistemas de mensagens JMS com mais funcionalidades como "JMS Bridging", alta disponibilidade e suporte para diversas linguagens de programação como Java, C/C++, C#, Ruby, Phyton e PHP.

Na Tabela 2.1 são comparadas as especificações CORBA (ORB), JMS (MOM), OGSI e WS-*.

Tabela 2.1: Comparação das diversas especificações em sistemas de notificações por eventos

	CORBA Event Service	CORBA Notification Service	JMS	OGSI-Notification	WS-Notification	WS-Eventing
Primeira Edição	3/1995	6/1997	1998	6/27/2003	1/20/2004	1/7/2004
Última Edição	10/2/2004	10/11/2004	4/12/2002	6/27/2003	2/2006	8/30/2004
Criador(es)	OMG	OMG	Sun Microsystems	Global Grid Forum	IBM, Sonic, TIBCO, Akami, SAP, CA, HP, Fujitsu, Globus	IBM, BEA, CA, Sun, Microsoft, TIBCO
Transporte	RPC	RPC	JMS	HTTP	Independente	Independente
Intermediário	Canal de Eventos	Canal de Eventos	Queue, Tópico	directamente, intermediário	directamente, intermediário	directamente, intermediário
Modo de Entrega	Push, Pull, Ambos	Push, Pull, Ambos	Push, Pull	Push	Push, Pull	Push por defeito, Pull e outros
Estrutura da Mensagem	Genérica, Tipada	Genérica, Tipada Estruturada Sequência de estruturas	TextMessage BytesMessage MapMessage StreamMessage ObjectMessage	SOAP	SOAP	SOAP
Filtro	-	Canal de Eventos	Nome da Queue/Topic	Nome de Serviço de dados	expressão lógica	expressão lógica
QOS	-	13 propriedades QOS	Prioridade, Persistência, Durabilidade, Transacções, Ordem das mensagens	-	Depende de outras especificações WS-*	Depende de outras especificações WS-*

A Tabela 2.1 permite tirar conclusões acerca da evolução das arquitecturas Event-Driven ao longo do tempo. Desta forma, é notório que o tipo de transporte tem vindo tornar-se independente, as mensagens têm tendência a usar estruturas que facilitem a interoperabilidade como SOAP através de XML, os filtros de mensagens estão a evoluir dos modelos baseados no nomes dos tópicos para modelos baseados em expressões que correspondam ao conteúdo das mensagens e a qualidade de serviço é cada vez mais implementada de uma forma modular. Verifica-se, portanto, que as novas soluções têm permitido o aumento da interoperabilidade entre produtores, consumidores e intermediários de mensagens.

Nas secções seguintes são apresentadas alternativas para MOMs, Advanced Messaging Queuing Protocol [7], Streaming Text Oriented Messaging Protocol [9] e OpenWire [10] que visam sobretudo resolver as limitações de interoperabilidade existentes em JMS.

Advanced Messaging Queuing Protocol

Advanced Messaging Queuing Protocol (AMQP) [7] é um padrão de comunicação de mensagens que opera ao nível da rede, isto é, um protocolo ao nível binário que usa um fluxo de bytes, ou "streams de bytes" para comunicar, normalmente por TCP. Desta forma, este protocolo pode ser usado em linguagens de programação que consigam interpretar a stream da mensagem. Os principais componentes do modelo de comunicação AMQP são as "exchanges" e as "queues" de eventos. O "exchange" baseado num conjunto de configurações tem a responsabilidade de decidir para que queue vai encaminhar o evento recebido. A "queue" mensagens procede à persistência dos eventos até que possam ser entregues aos subscritores. Este encaminhamento pode ser configurado para suportar diferentes topologias, como ponto-a-ponto ou de publicação/subscrição.

Este protocolo começou por ser usado na área financeira onde alto desempenho, escalabilidade, fiabilidade e segurança são requisitos fundamentais. Actualmente, existem diversas plataformas como Apache QPid, RabbitMQ e OpenAMQ que implementam este protocolo para desenvolvimento de serviços de mensagem.

A especificação do protocolo AMQP ainda não está terminada, começou em 2006 com a versão 0-8 e entretanto já foram publicadas as versões 0-9, 0-10 e 0-9-1 [14].

Streaming Text Oriented Messaging Protocol

Streaming Text Oriented Messaging Protocol (STOMP) [9] é um protocolo direccionado à interoperabilidade entre linguagens de programação, plataformas e intermediários de mensagens. Este protocolo destaca-se devido à sua simplicidade e facilidade de implementação do lado do cliente para o envio de mensagens, existindo actualmente implementações em C, Dynamic C, C++, C# .NET, Delphi, FreePascal, Earlang, Flash, Java, Objective-C, Perl, PHP, Pike, Python, Ruby e Smaltalk.

Este protocolo opera ao nível de rede com as instruções SEND, SUBSCRIBE, UNSUBSCRIBE, BEGIN, COMMIT, ABORT, ACK e DISCONNECT, onde cada instrução possui um cabeçalho e um corpo de mensagem.

A implementação deste protocolo por parte dos intermediários de mensagens não é tão simples visto normalmente possuírem diversos protocolos de mensagem, tendo que, por conseguinte proceder ao mapeamento das instruções entre protocolos. Actualmente, plataformas como Apache ActiveMQ e HornetQ suportam o processamento de mensagens STOMP.

OpenWire

OpenWire [10] é o protocolo de rede usado por omissão pelo intermediário de mensagens Apache ActiveMQ. Este protocolo permite acesso nativo ao intermediário através de diferentes linguagens e plataformas. A plataforma Apache ActiveMQ fornece uma API para OpenWire para Java, C, C++ através do sub-projecto Apache ActiveMQ CMS[1] e C# através do sub-projecto Apache ActiveMQ NMS[2].

2.1.3 Staged Event-Driven Architecture

As Staged Event-Driven Architecture (SEDA) [15] consistem numa rede de estágios Event-Driven que permitem o processamento de eventos com altos níveis de concorrência, visto adequarem a carga existente à disponibilidade dos serviços. Em sistemas assíncronos, é comum o envio de eventos com uma frequência superior à capacidade de processamento suportada, com as SEDA evitam-se tentativas de uso de mais recursos do que os disponíveis. Desta forma, a SEDA decompõe uma aplicação numa rede de estágios separadas por queues de eventos, cada

[1] C++ Messaging Service
[2] .Net Messaging Service

estágio é constituído por um processador de eventos, uma queue que recebe os eventos, uma pool de threads e um controlador responsável por agendar e alocar threads. Cada thread é responsável por obter um grupo de eventos da queue e invocar o processador. Depois de processados os eventos e de acordo com a lógica de cada estágio podem ser enviados zero ou mais eventos a outros estágios. O nível de concorrência é da responsabilidade do controlador que deve de forma dinâmica ajustar o número de threads de cada estágio. As principais vantagens existentes numa SEDA, são as seguintes [15]:

- Suporte a altos níveis de concorrência: sempre que possível, a SEDA utiliza execuções event-driven em vez de threads, para evitar perda de performance.
- Simplifica a construção de serviços facilmente adaptáveis: de forma a reduzir a complexidade no desenvolvimento de serviços, a SEDA esconde dos programadores muitos detalhes de agendamento e gestão de recursos. A estrutura desta arquitectura facilita o desenvolvimento modular e fornece suporte de análise de desempenho.
- Permite análise interna: as aplicações podem analisar os pedidos e adaptar o seu comportamento às condições de carga.
- Suporte para auto ajuste da gestão de recursos: em vez de definir os requisitos de recursos das aplicações e carga dos clientes, o sistema ajusta-se dinamicamente aos requisitos de performance.

No contexto de monitorização de aplicações as SEDA permitem uma melhor gestão dos recursos utilizados na comunicação com os componentes monitorizados e no processamento de eventos recebidos, prevenindo que em situações de maior carga a arquitectura fique indisponível.

2.1.4 Processamento de Eventos

Existem três tipos de processamento de eventos [16]: simples, fluxo (“stream”) e complexo.

O processamento de eventos simples está associado a acções únicas que dependem da lógica de negócio, reduzindo atrasos e custos no negócio.

O processamento de eventos de fluxo envolve a troca de informação por vários sistemas de negócio, cada qual com uma lógica própria, permitindo a ocorrência de decisões imediatas.

O processamento de eventos complexos (CEP) baseia-se na avaliação de um conjunto de vários eventos para posteriormente realizar acções. CEP requer interpretadores sofisticados, a

definição e correspondência de padrões e técnicas de correlação, estando normalmente associado à detecção de anomalias no negócio, ameaças e oportunidades.
As lógicas de negócio das aplicações são cada vez mais integradas, desta forma os sistemas de análise estão a evoluir no sentido do processamento de cargas elevadas de eventos [17], esperando-se um grande crescimento de aplicações nesta área. Este tipo de processamento é cada vez mais usado na gestão de processos de negócio (BPM) como forma de rentabilização da gestão de processos, gerando alertas com oportunidades de negócio através da detecção de um conjunto de correlações de eventos importantes [18]. Outra área de aplicação são os dispositivos sensoriais como RFID que permitem processar múltiplos eventos com a finalidade de detectar padrões complexos [19].

As técnicas de processamento de eventos são portanto fundamentais em arquitecturas orientadas a eventos, podendo ser utilizados em sistemas de monitorização, utilizando o CEP como forma de prever possíveis situações anómalas.

2.1.5 Intermediário de Mensagens

Um intermediário de mensagens é um componente responsável por gerir a comunicação entre aplicações distintas. As aplicações deixam de comunicar directamente, evitando que as aplicações se tenham de conhecer, desta forma, comunicam unicamente com um intermediário de mensagens. O intermediário tem a vantagem de poder disponibilizar diferentes interfaces de comunicação, podendo gerir o encaminhamento tendo em conta diversos aspectos que melhoram e asseguram a comunicação:

- Clustering: Envolve os processos de descoberta de serviços e intermediários, sistemas de failover, balanceamento de carga, qualidade de serviço, alta-disponibilidade, entre outros.

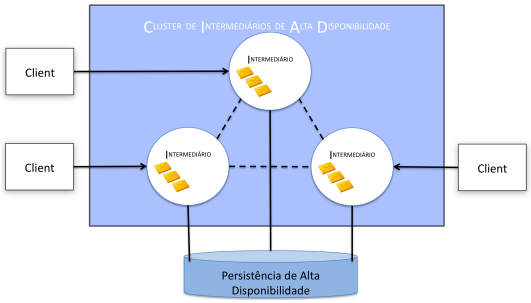

Figura 2.3: Cluster de Intermediários de Alta Disponibilidade

- Suporte para diversas linguagens e protocolos de mensagens: Para além da linguagem e protocolos de mensagens nativos suportados pelo intermediário, estes sistemas são implementados para comunicar através de protocolos como Web Services, AMQP e JMS que disponibilizam APIs para diversas linguagens.

- Suporte para diversos protocolos de transporte: A comunicação com o intermediário pode ser efectuada através de diferentes protocolos de transporte, tal como, in-VM para comunicações efectuados por aplicações em execução na mesma JVM do intermediário, UDP, TCP, NIO, SSL, multicast, entre outros.

- Persistência de mensagens: As mensagens recebidas podem ser persistidas, por exemplo em base de dados, para garantir que a mensagem nunca é perdida independentemente do consumidor estar indisponível ou de existir alguma falha na entrega da mensagem. De forma a maximizar a eficiência e fiabilidade do processo de persistência das mensagens são usadas caches e sistemas de log ("journaling").

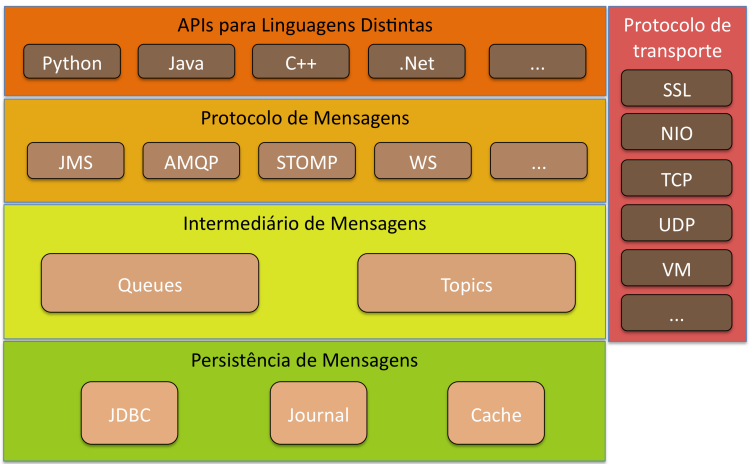

Figura 2.4: Comunicação com o Intermediário de Mensagens

- Divisão de mensagens por destino: Podem ser configurados filtros e rotas de encaminhamento para que as mensagens possam ser replicadas ou desviadas.

- Enterprise Application Integration: Através de padrões de integração existem mais protocolos e tecnologias de transporte suportadas facilitando o encaminhamento e transformação de mensagens.

- Ligação do intermediário para servidores com o mesmo protocolo: Permite configurar os intermediários de mensagens para que todas as mensagens recebidas para uma determi-

nada queue ou tópico sejam encaminhadas para um servidor que suporte o mesmo protocolo. Por exemplo, se o intermediário suportar mensagens JMS pode ser configurada uma "bridge" com o serviço de mensagens JMS de alta performance do servidor aplicacional JBoss [20].

- Controlo de fluxo: De forma a garantir que os limites de memória no intermediário não são excedidos, nas ligações síncronas é abrandado o fluxo dos produtores de mensagens atrasando o envio de confirmação da recepção da mensagem, enquanto que em ligações assíncronas o produtor deve especificar um tamanho máximo para a janela de envio.

Actualmente, existem um conjunto de implementações de intermediários de mensagens que se dividem em dois subconjuntos: código aberto e proprietário.

Dentro dos proprietários destacam-se IBM Websphere, TIBCO Enterprise Message Service e Oracle WebLogic Server. Em código aberto existem soluções como ActiveMQ [21], HornetQ [22], OpenAMQ [23], OpenMQ [24], Qpid [25] e RabbitMQ [26], cujas funcionalidades são apresentadas na Tabela 2.2.

Tabela 2.2: Comparação das principais funcionalidades de intermediários de mensagens

	ActiveMQ	HornetQ	OpenAMQ	OpenMQ	Qpid	RabbitMQ
Linguagem de Implementação	Java	Java	C	Java	Java e C++	Erlang
Protocolo de Mensagens	JMS 1.1	JMS 1.1	AMQP/0.9.1	JMS 1.1	AMQP/0.10	AMQP/0.8/0.9
Protocolos Suportados	OpenWire, REST, Stomp, WS Notification, XMPP	Stomp REST Web Sockets	-	Stomp	-	Stomp, XMPP
Linguagens Suportadas	**Java**, Clientes Stomp, **C/C++** (Stomp, OpenWire e CMS[1]), **Ajax** (Rest), **WebSockets**, **C#** e **.NET** (NMS[3]), **Delphy/FreePascal**, (Habari) **JavaScript** (Ajax e WebSockets), etc.	Java, Clientes Stomp	C e aplicações de terceiros para Python, Java (JMS) e Ruby	Java e Clientes Stomp	C++, Java (JMS), Python, Ruby, C# .NET, Adaptador WCF[2] e Ruby	C, Erlang e clientes stomp
Clustering	descoberta failover balanceamento replicação	descoberta failover balanceamento replicação	failover balanceamento (só de subscritores)	descoberta failover balanceamento replicação	descoberta failover replicação	descoberta failover balanceamento replicação
Protocolos relevantes	in-VM SSL (2-way)	in-VM SSL (2-way)	-	in-VM SSL (1-way)	in-VM SSL (2-way)	SSL (2-way)
Bridging	Sim	Sim	Não	Sim	Não	Não
Persistência	Sim	Sim	Não	Sim	Sim	Sim
Controlo de Fluxo	Sim	Sim	Sim	Sim	Sim	Sim
EAI	Apache Camel	third-parties	-	-	-	-

2.2 Integração Aplicacional

Enterprise Application Integration (EAI) [27, 28] designa a integração de aplicações, sistemas e processos para sistemas de negócio que necessitem uma rápida integração de serviços. Este conceito baseia-se em diferentes padrões de integração, normalmente divididos em integração de dados disponibilizando dados de várias origens, integração de objectos de forma a haver uma cooperação entre objectos, integração de funções e métodos para partilha da lógica de negócio, integração de interfaces das aplicações para integração entre aplicações diferentes e integração de processos.

Desta forma, a monitorização de múltiplos sistemas, com subcomponentes heterogéneos em contextos empresariais, pode tirar proveito deste conceito de interoperabilidade através do uso de padrões EAI para implementar um solução eficiente.

2.2.1 Estratégias existentes

Existem diferentes estratégias que permitem a implementação do conceito de integração, como através adaptadores, servidores aplicacionais, intermediários e através da gestão de processos do negócio [29]:

Adaptadores

Os adaptadores resolvem a falta de interoperabilidade entre diferentes tecnologias de aplicações empresariais, fornecendo conectividade e tradução de pedidos num formato que a aplicação destino consiga processar. Normalmente, esta tradução é realizada com dois adaptadores, um da origem para um formato genérico de dados, protocolos, entre outros e outro deste formato genérico para um formato que a aplicação destino possa processar. O formato genérico dos dados são padrões que facilitam a tradução, tal como, XML, IDL ou interfaces como WSDL, que também é baseado em XML e usado nos Web Services [29].

[1] ActiveMQ C++ Messaging Service
[2] Windows Communication Foundation
[3] ActiveMQ .Net Messaging Service

Servidores Aplicacionais

Os servidores aplicacionais foram criados tendo em vista as aplicações web, permitindo a separação entre a camada lógica de negócio e a camada de apresentação e disponibilizando uma solução coordenada de ligações aos recursos. Para além da separação das camadas aplicacionais, permite no nível lógico a integração de processos de negócio assegurando funcionalidades de comunicações transaccionais, desempenho, "clustering" e disponibilidade. Estes conceitos de integração estão presentes em plataformas como Java EE, JBoss, .NET e WebSphere [29].

Intermediários de integração

A integração por intermediários permite a comunicação de diversas aplicações, garantindo segurança, encaminhamento e transformação de dados através de adaptadores. Os intermediários de integração foram inicialmente baseados em intermediários "hub and spoke" [28] que usa um servidor centralizado onde as diferentes aplicações se conectam para enviar e receber dados. Os intermediários de integração são capazes de partilhar informação com vários sistemas através de mecanismos event-driven assíncronos. Na Figura 2.5 são apresentados os elementos normalmente associados a um intermediário "hub and spoke", ficando claro que o intermediário de integração é o componente central do sistema, facilitando a troca de informação entre as várias aplicações. Os adaptadores permitem uma divisão clara entre as camadas com a API das aplicações e a infra-estrutura de mensagens, facilitando a integração das aplicações num sistema com componentes desassociados.

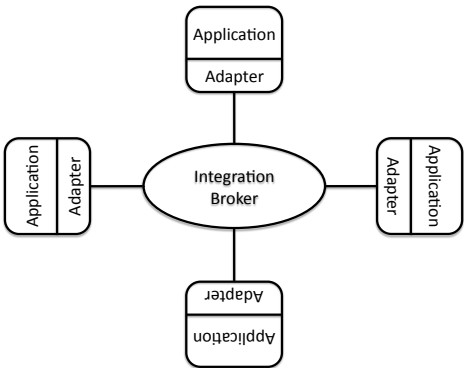

Figura 2.5: Intermediário "hub and spoke"

Os sistemas de mensagens do tipo MOM, baseiam-se no paradigma "hub and spoke" pelo que muitos destas plataformas oferecem facilidades de integração, portabilidade e flexibilidade,

normalmente possuem uma fila lógica associada para persistir as mensagens recebidas, e para as encaminhar assim que o destino estiver disponível. Para além dos sistemas MOM, existem barramentos de integração, onde as aplicações utilizam adaptadores para comunicarem com o canal. Deste modo, são reduzidas as ligações ponto-a-ponto disponibilizando uma arquitectura semelhante à de eventos para comunicação entre aplicações heterogéneas [29].

Business Process Management

Em empresas que fornecem múltiplos serviços a clientes, o conceito de gestão dos processos de negócio (BPM) permite obter controlo sobre todos os processos e interacção humana para estes serviços. Deste modo, sistemas orientados ao processo de negócio oferecem um sistema de negócio mais eficiente e com facilidade de integração para dados, processos autónomos de negócio, análise de processos e níveis de visualização.

Os processos de negócio devem ser automatizados para controlar e coordenar as actividades com ferramentas que definem quer o fluxo da informação entre as aplicações, quer o fluxo entre processos de interacção humana [29].

2.2.2 Enterprise Service Bus

Enterprise Service Bus (ESB) é um padrão em barramentos de mensagens que inclui as quatro estratégias de integração descritas atrás. Com este padrão é possível implementar, instalar e gerir arquitecturas SOA, disponibilizando processamento distribuído e padrões de integração como adaptadores e interfaces. As funcionalidades de comunicação são baseadas em modelos empresariais de mensagens que estão disponíveis no barramento, estes modelos definem o formato das mensagens que o ESB usa para receber e transmitir. As funcionalidades principais de um ESB são a orquestração de serviços permitindo ter diferentes serviços agregados, a definição do fluxo da informação dos processos de negócio, encaminhamento de eventos, qualidade dos serviços como segurança, políticas, fiabilidade e adaptadores de tradução. ESB está normalmente relacionado com SOA, visto que permite a implementação de aplicações como serviços, disponibiliza orquestração de serviços e facilidade de gestão de serviços [29].

Na Figura 2.6 são apresentados os elementos normalmente associados a um ESB, os adaptadores, o barramento, aplicações cliente e os serviços.

As aplicações de nível empresarial têm em conta que a lógica de fluxo de negócio deve suportar processos assíncronos, isto é, o fluxo dos processos deverá contemplar situações que po-

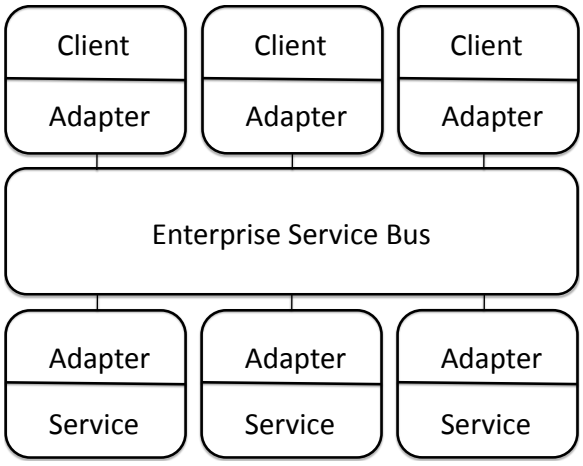

Figura 2.6: Enterprise Service Bus

dem ocorrer assincronamente, desta forma com ESB pode-se criar uma arquitectura que englobe EDA e SOA. Com arquitecturas EDA e SOA são disponibilizados serviços assíncronos a um barramento, desta forma as funcionalidades de negócio permanecem disponíveis mas de forma desassociada através de eventos. Os publicadores são os adaptadores que recebem pedidos e os enviam aos serviços que fizeram subscrição no barramento. As principais funcionalidades dos ESB são:

- Funcionalidades do Serviço de Comunicação: Permite transferir e encaminhar mensagens de um protocolo para outro diferente suportando modelos de mensagens de para interfaces SOA.
- Conectividade dinâmica de serviços: Disponibiliza uma API para conexões dinâmicas independentes da implementação de protocolo usado.
- Funcionalidades de encaminhamento por tópico ou conteúdo: Os mecanismos de encaminhamento podem ser baseados no nome dos tópicos ou no conteúdo dos eventos. O encaminhamento por tópicos baseia-se na criação de tópicos diferentes, permitindo definir grupos de subscrições por tópico. Encaminhamento baseado no conteúdo permite publicar mensagens a determinados subscritores de acordo com propriedades da mensagem, este tipo pode requerer o uso de XML.
- Descoberta do destino: permite a escolha do melhor destino em tempo real, desta forma, o destino pode ser escolhido de acordo com a qualidade dos serviços de um destino ESB, usando um serviço apropriado como UDDI.

- Integração: ESB pode ser configurado com diferentes tipos de padrões de integração, disponibilizando diferentes tipos de interoperabilidade.

- Transformação: A transformação do formato de uma mensagem é muito importante nos ESB, porque permite que aplicações heterogéneas se conectem ao barramento para receber dados no formato esperado.

- Fiabilidade: Garante que as mensagens publicadas no barramento são devidamente entregues. Deste modo, as mensagens são persistidas até que o subscritor confirme.

- Segurança: Disponibiliza um modelo de segurança para encriptação, autenticação, autorização, entre outras como também para integração de diferentes modelos de segurança.

- Transacções e serviços duráveis: Existem serviços que devem suportar uma conexão ou conversação contínua. Os ESBs garantem uma subscrição permanente e também assegura q recuperação de falhas através de um serviço transaccional.

- Gestão e monitorização: Numa infra-estrutura ESB a gestão deve suportar diversas configurações diferentes de serviços e deve garantir a integração com sistemas heterogéneos. A monitorização é essencial para assegurar a qualidade do serviço estipulada. Este processo é realizado usando o barramento para aceder a estatísticas e métricas do serviço.

- Escalabilidade: Com a rápida de integração de novos serviços é essencial que as funcionalidades de um ESB sejam facilmente escaláveis, isto é, um ESB deve disponibilizar um número indefinido de serviços.

2.2.3 Protocolos de serialização de mensagens

A serialização consiste em converter instâncias de determinadas estruturas de mensagens em sequências de bits. A troca de mensagens serializadas permite a comunicação entre plataformas com diferentes linguagens de programação, o uso de mecanismos de encriptação e compressão e uma fácil adaptação a alterações na estrutura das mensagens. Existem actualmente diferentes mecanismos de serialização, cujas principais diferenças se baseiam nas linguagens de programação suportadas, no tamanho da informação serializada, nos tempos de serialização/desserialização e de criação dos objectos. De seguida os protocolos existentes são comparados de forma a avaliar qual o mais adequado:

Abstract Syntax Notation One

ASN.1 é uma notação para definir a sintaxe de dados de informação e que não restringe a forma como a informação é codificada para transmissão [30]. Actualmente, existem diferentes regras de codificação, que permitem a troca de informação quer entre aplicações implementadas com diferentes tecnologias quer entre máquinas heterogéneas:

- Basic Encoding Rules: BER é conjunto de regras de codificação/descodificação que podem ser usadas em informação especificada segundo a notação ASN.1. Os dados são codificados em octetos com uma estrutura baseada em quatro componentes, que identificam o tipo, tamanho, conteúdo e o fim do conteúdo. A indicação com o fim do conteúdo é apenas necessária quando o tamanho é indefinido, permitindo que a codificação possa usar apenas os primeiros três componentes [31].

- Canonical Encoding Rules: CER é um subconjunto das regras BER, a principal restrição é a forma como é definido o tamanho de um tipo de valor codificado. Caso seja um tipo primitivo deve ser definido o menor número de octetos de comprimento necessários, caso seja um tipo construído a partir de outros tipos primitivos o comprimento deve ser especificado como indefinido [31].

- Distinguished Encoding Rules: DER, tal como CER, é um subconjunto das regras BER, onde o componente com o tamanho deve ser definido sempre com o menor número de octetos [31].

- Packed Encoding Rules: PER define um conjunto de regras de codificação para a informação representada segundo a notação ASN.1, permitindo uma representação mais compacta do que as BER [32].

- XML Encoding Rules: XER permite uma codificação baseada em XML. O XER possui dois subconjuntos de regras de codificação, Basic XML Enconding Rules e Canonical XML Enconding Rules [33].

- Generic String Encoding Rules: GSER permite a codificação para um formato de texto legível por pessoas. O conjunto de regras desta norma foram definidas para o LDAP (Lightweight Directory Access Protocol), no entanto podem ser utilizadas noutras áreas [34].

External Data Representation

XDR é um standard para a representação e codificação de informação, permitindo a comunicação entre plataformas com arquitecturas distintas. Apresenta-se de uma forma mais simples que o ASN.1, visto que os tipos de dados usados são conhecidos por ambas as partes, no entanto torna-se menos flexível. Este protocolo é usado pelos sistemas ONC RPC (Open Network Computing Remote Procedure Call) e NFS* (Network File System) para representar a estrutura da informação [35].

Protocol Buffers

Desenvolvido pela Google, permite a codificação de informação especificada a partir de um formato próprio. A Google disponibiliza APIs para C++, Java e Python, no entanto, existem aplicações de terceiros para mais linguagens como C#, Erlang, Haskell, Javascript, Perl, PHP, Ruby, entre outras. Este protocolo é usado pela própria Google em protocolos RPC e formatos de ficheiro [36].

Thrift

Thrift permite através da especificação de estruturas de dados e serviços a codificação de informação e a criação de plataformas RPC em várias linguagens, como C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk e OCaml. Thrift foi desenvolvido inicialmente pelo Facebook, sendo actualmente um projecto open source da Apache [37].

Apache Avro

Apache Avro baseia-se em JavaScript Object Notation (JSON) para a serialização de mensagens e uso de protocolos RPC. Este sistema permite a geração de código para aplicações que usem sempre uma estrutura de dados específica ou o uso de estruturas de dados genéricas que apenas são conhecidas em tempo de execução. Este sistema difere do Thrift e Protocol Buffers visto não implicar a geração de código sempre que se altera a estrutura de dados. O facto de as estruturas de dados serem definidas em JSON facilita a implementação em linguagens que suportem esta notação, no entanto, o sistema Apache Avro fornece APIs para Java, C, C++, Python e Ruby. Para facilitar a definição da estrutura da informação em JSON existe uma linguagem própria de Apache Avro (GenAvro Language), actualmente em fase experimental que gera uma estrutura de

dados JSON a partir de uma linguagem de alto-nível de forma semelhante ao que acontece com o Google Protocol Buffers e o Thrift [38].

Serialização para plataformas Java

Para além dos sistemas de serialização apresentados existe outro conjunto que não possui APIs para linguagens de programação distintas. Destaca-se em especial o Java através da interface java.io.Externalizable, da framework Kryo da Google, Jackson Java JSON-processor e Aalto Stax XML Processor.

Na Tabela 2.3 é apresentada uma comparação dos protocolos de serialização que possuem APIs em linguagens diferentes. De forma a comparar os tempos de serialização, desserialização, criação de objectos e tamanho da informação serializada são apresentados os resultados de um benchmark inicialmente criado no Google Code [39] e posteriormente movido para o github [40]. As normas ASN.1 e XDR não são comparados nesta tabela visto serem apenas standards de mensagens. No entanto, o seu uso no âmbito desta dissertação será avaliada de forma a decidir qual a melhor opção.

Tabela 2.3: Comparação de protocolos de serialização

	Linguagens Suportadas	Tempo de Serialização (nanosegundos)	Tempo de Desserialização (nanosegundos)	Tempo de Criação de Objectos (nanosegundos)	Tamanho da mensagem (bytes)
Protocol Buffers	C++, Java, Python e third-party	4115	2428	482	239
Thrift	C++, Java, Python, Ruby, Erlang, Perl, C# PHP, Haskell, Cocoa, Smalltalk e OCaml	4139	4609	408	349
Apache Avro (generic)	Java, C, C++, Python e Ruby	5226	5557	2380	221
Apache Avro (specific)	Java, C, C++, Python e Ruby	4674	6065	1857	221

2.3 Sumário

Neste capítulo foi apresentada toda a informação necessária para dar início ao trabalho a desenvolver, caracterizando-se por ser um capítulo muito teórico mas essencial para o entendimento das arquitecturas Event-Driven. Como tal, o estado da arte incidiu na discussão da forma lógica como os eventos são processados, apresentando as plataformas de mensagens existentes e tecnologias actuais para intermediários de mensagens. Tendo em conta os diversos serviços e subsistemas da solução NGIN, foram apresentadas para as plataformas de intermediação de

eventos o estado da arte de padrões e protocolos de integração, nomeadamente protocolos de serialização de mensagens e protocolos de transporte que facilitam a independência de plataformas na comunicação.

As abordagens apresentadas neste capítulo assumem assim, um papel fulcral para o início do desenvolvimento da solução, discutida no próximo capítulo.

Capítulo 3

Concepção e implementação da solução

Após a contextualização do tema proposto nesta dissertação, serão apresentadas neste capítulo, todas as etapas, pelas quais passou o desenvolvimento da solução final. Inicialmente será abordada a solução actual, usada na PT Inovação, para monitorização de aplicações e subsistemas NGIN, o subsistema NGIN Manager, sendo posteriormente demonstrado a forma como a solução desenvolvida evolui a solução actual. Deste modo, este capítulo apresenta os requisitos da arquitectura orientada a eventos e os detalhes de implementação de cada componente da nova solução.

3.1 A solução actual do NGIN Manager na PT Inovação

A solução NGIN é uma solução extremamente complexa, com um elevado número de máquinas e aplicações, o que torna a operação e suporte complicadas para se efectuar com base na monitorização directa das aplicações e máquinas.

O NGIN Manager é um OSS (Operations Support Systems), desenvolvido para simplificar as tarefas de operação e suporte das plataformas NGIN. Normalmente estas ferramentas são críticas para os operadores de telecomunicações dado que necessitam, a cada instante, de saber quais os problemas que existem na sua infra-estrutura. Falhas na detecção de problemas ou no consequente processamento pode originar perdas de receitas ou insatisfação e desagrado dos seus clientes (pode, inclusive, culminar com a perda do cliente para uma operadora concorrente).

Actualmente, o produto NGIN Manager é responsável pela monitorização de componentes da rede inteligente e subsistemas da solução NGIN através do cálculo de métricas de desempenho e geração de alarmes em casos de falhas ou em casos de se exceder os limites definidos para as

métricas calculadas. O NGIN Manager procede à monitorização da execução de serviços de rede inteligente nos SCFs, DSCFs, SRFs através de ficheiros e nos SDFs através de MIBs próprias para onde é enviada a informação calculada usando o protocolo SNMP, os alarmes são enviados através de traps SNMP. A monitorização dos subsistemas da solução NGIN é efectuada ao nível da execução de operações, para sistemas que operam em base de dados como ETT (Event Ticket Transfer), BREE (Business Rules Execution Environment) ou LTR (Lost Ticket Recovery) são usados procedimentos em PL/SQL que permitem armazenar, em tabelas específicas, informação referentes ao processo de monitorização. Os sistemas com módulos que operam ao nível do sistema operativo como NRF (NGIN Rating Function) utilizam o protocolo SNMP para o envio de informação. Para além de procedimentos PL/SQL e de transferência por SNMP o NGIN Manager monitoriza as operações do sistema SMINT (Smart Interface) por JMX.

O processo de análise da informação recolhida requer um prévio registo das entidades monitorizadas por parte do back-end do NGIN Manager, posteriormente a informação é processada e catalogada numa data warehouse nas dimensões da entidade e tipo de métrica usada para que o front-end do NGIN Manager possa visualizar o inventário, métricas de desempenho e alarmes. Os alarmes para além de serem visualizados no front-end podem ser configurados para serem enviados por sms e/ou email.

Para além dos módulos responsáveis pela gestão do inventário das entidades monitorizadas, gestão das métricas de desempenho e gestão de alarmes o NGIN Manager contempla a visualização de estatísticas de monitorização em tempo real, recorrendo à apresentação de gráficos com os valores recolhidos em applets Java. Na Figura 3.1 é apresentada a visão funcional da arquitectura lógica do NGIN Manager.

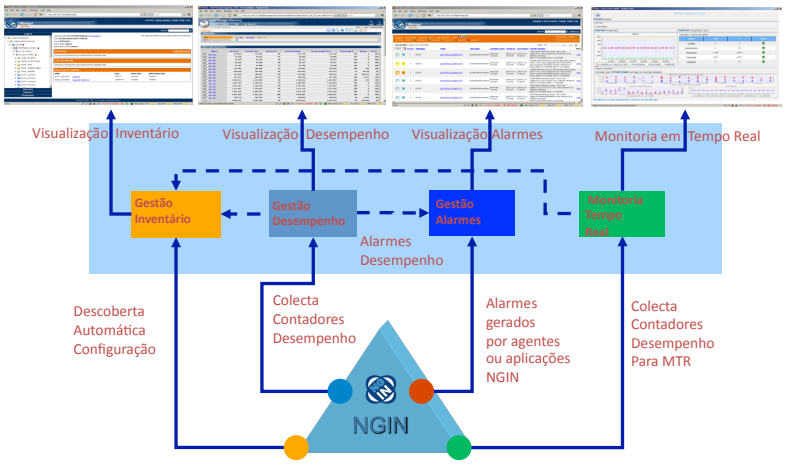

Figura 3.1: Arquitectura lógica do NGIN Manager

3.2 Requisitos da Solução

A solução OSS NGIN Manager do sistema NGIN tem evoluído integrando cada vez mais subsistemas da solução e componentes da IN, no entanto a solução actual é sujeita a falhas e a perda da informação nos processos de transferência. No caso da comunicação por transferência de ficheiros para o NGIN Manager é usado FTP ou SSH, posteriormente os ficheiros são processados e são obtidas métricas específicas. Para esta solução existem diversos problemas como elevado número de ligações FTP/SSH, ocupação da largura de banda, falhas de rede que leva à perda de informação e erros de parsing do ficheiro devido a alterações no formato dos relatórios. Para o caso da comunicação por SNMP existe pouca fiabilidade dado que, usa UDP como protocolo de transporte, possui pouco desempenho na persistência da informação, não é um sistema escalável e não oferece facilidade no desenvolvimento de funcionalidades que melhorem a gestão. A transmissão de informação para subsistemas da solução NGIN que operam ao nível de base de dados é efectuada através de procedimentos PL/SQL que armazenam informação de monitorização em tabelas específicas, esta solução tem como problemas o facto de o NGIN Manager ter que efectuar, para cada entidade monitorizada, pooling ao conteúdo das tabelas, forçando, consequentemente à existência de sincronização entre o NGIN Manager e os processos de monitorização. No caso da comunicação por JMX, existe a limitação de ser uma solução unicamente orientada para plataformas Java.

Desta forma, surgiu a necessidade de criar uma nova plataforma de monitorização, com os seguintes requisitos:

- A nova solução deverá oferecer fiabilidade na comunicação entre os componentes monitorizados e o NGIN Manager. De forma a superar falhas e perdas de informação deve garantir que a informação é persistida, efectuando a gestão do processo de envio de notificações de acordo com as condições dos vários elementos envolvidos na comunicação.

- Pretende-se que a nova plataforma tenha em conta a necessidade de uma disponibilidade constante, isto é, não é aceitável que o processo de envio de notificações de monitorização se possa atrasar devido à indisponibilidade do serviço de envio de notificações.

- O desempenho da plataforma é um requisito essencial para garantir que o envio de notificações é realizado o mais rapidamente possível, isto é, o processo de persistência das notificações, consequente envio e confirmação da recepção devem possuir mecanismos de alto desempenho de forma a suportar cargas elevadas.

- A nova solução de monitorização deverá possibilitar uma plataforma escalável, permitindo assegurar requisitos de desempenho e disponibilidade.

- Devido à possível heterogeneidade das máquinas monitorizadas, espera-se que a plataforma possua APIs para arquitecturas distintas.

- É esperado que a plataforma tenha pouco impacto nos agentes responsáveis por recolher informação quer na interacção dos agentes com a nova plataforma, quer na ocupação de recursos de memória e de rede.

- Pretende-se a existência de flexibilidade/sistematização na estrutura das notificações, de forma a que possa ser usada por diferentes aplicações com o mínimo de alterações possíveis.

- Dado o impacto que a informação da notificação pode ter é necessário que a mensagem possa ser codificada antes de ser enviada e que a plataforma possua mecanismos de autenticação.

3.3 Desenho da Arquitectura

Perante a solução actual do NGIN Manager fica evidente a necessidade da nova arquitectura como forma de assegurar a fiabilidade na comunicação com os vários componentes da solução NGIN. De acordo com os requisitos propostos, a nova arquitectura irá possibilitar uma uniformização do processo de envio de notificações, facilitando consequentemente a integração de novos componentes no processo de monitorização do NGIN Manager.

Nesta secção será apresentada a nova arquitectura, os respectivos componentes, funcionalidades e decisões tomadas na escolha das tecnologias. Será, de igual forma, descrito como os vários componentes da solução vão comunicar apresentando a estrutura de mensagens usada para o envio de notificações. De forma a ter em consideração o impacto da nova arquitectura, será feito um enquadramento da actual solução no NGIN Manager, apresentando os principais componentes monitorizados na rede inteligente e subcomponentes da solução NGIN, descrevendo os respectivos mecanismos de envio da informação e as melhorias conseguidas com a nova arquitectura.

3.3.1 Enquadramento da nova arquitectura no NGIN Manager

A integração da arquitectura orientada a eventos no NGIN Manager implica que por um lado se continue a suportar monitorização ao nível infra-estrutural, de execução de serviços e execução de operações e por outro lado que os componentes actualmente monitorizados possam usar a nova solução, tal como apresentado na Figura 3.2. O suporte de diferentes níveis de monitorização implica apenas que a estrutura de mensagens seja suficientemente flexível para conseguir englobar diferentes tipos de informação, enquanto que a integração com os componentes actualmente monitorizados implica que estes sejam capazes de comunicar com a nova plataforma.

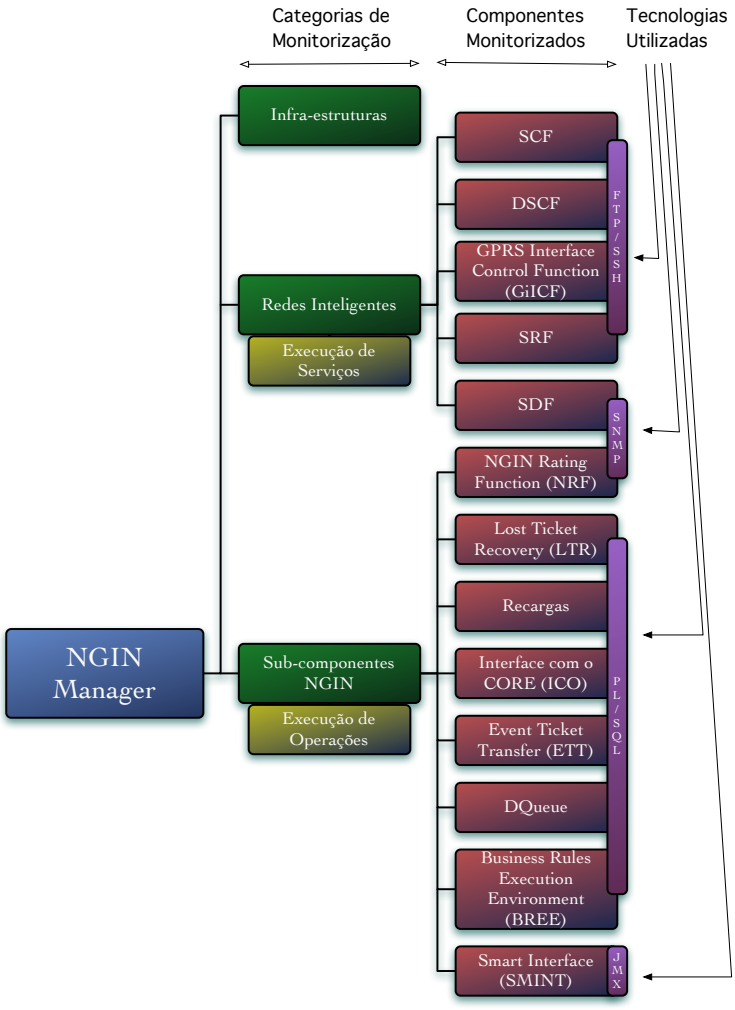

Figura 3.2: Componentes monitorizados pelo NGIN Manager

Como tal, para os componentes que actualmente transferem a informação de monitorização por FTP/SSH, SNMP e JMX é necessário, consoante o tipo de plataforma, criar uma API que permita a construção da mensagem e posterior envio, os componentes que usam procedimentos PL/SQL em base de dados Oracle para armazenar a informação de monitorização podem usar mecanismos que permitem a invocação de métodos JAVA a partir de funções PL/SQL.

A integração da arquitectura orientada a eventos com o NGIN Manager implica alterações no back-end, o front-end continua a ser usado sem alterações. A adaptação do back-end é necessária pela lógica da gestão de inventário, gestão de desempenho e monitorização em tempo real, dado que os componentes monitorizados enviam notificações quer com o valor das métricas monitorizadas como também com o inventário completo da máquina e descrição das entidades envolvidas. Desta forma, o processo de análise das notificações recebidas é simplificado, tendo em conta que o NGIN Manager possui um servidor aplicacional JBoss, bastando configurar um Message Driven Bean (MDB) [13] responsável por receber as notificações e adaptar a lógica de gestão de desempenho e inventário para descodificar e processar toda a informação recebida na notificação.

3.3.2 A Arquitectura da Solução

Como se pode ver na Figura 3.3, a nova arquitectura possui três entidades principais, isto é, os componentes responsáveis pela serialização e envio das notificações, o intermediário de mensagens e os componentes do NGIN Manager responsáveis por receber e processar notificações:

- **Envio de notificações**: Para o envio de notificações a lógica de monitorização existente é reaproveitada, sendo apenas necessário adaptá-la para usar a nova API que permite a serialização e envio de notificações.
  A dependência tecnológica da solução é mínima, dado que as ferramentas de serialização de mensagens geram estruturas de dados ou até mesmo código para determinadas linguagens de programação, permitindo que os protocolos de serialização, mensagens e transporte possam ser facilmente adaptados quer a alterações na estrutura da notificação, quer aos diferentes ambientes existentes.

- **Intermediação de eventos**: O intermediário de eventos é o ponto central da arquitectura que dá garantias de fiabilidade na comunicação através de mecanismos de persistência e garantias de disponibilidade e desempenho da solução através da facilidade de escalabilidade do sistema com mecanismos de clustering. Para além de garantias funcionais o

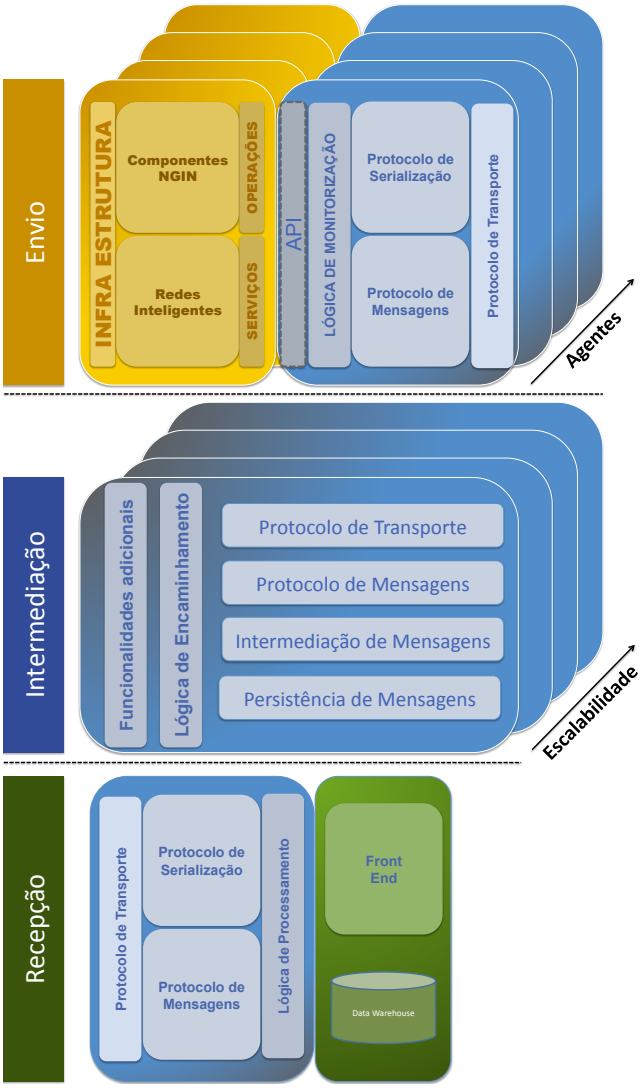

Figura 3.3: Arquitectura da solução

intermediário oferece garantias de integração, oferecendo por um lado, diferentes protocolos de mensagens, transporte e APIs para que os componentes responsáveis pelo envio e recepção de notificações possam escolher as tecnologias mais apropriadas ao seu ambiente de execução e por outro lado permitindo a implementação de novas funcionalidades. Em particular, foram implementadas funcionalidades para que o intermediário de eventos possa ser controlado remotamente e um plugin que permita a recepção de notificações por socket como extensão aos protocolos já existentes. Como será descrito em maior detalhe na secção seguinte o controlo do intermediário por outras aplicações permite não só arrancar e parar a aplicação, como também o arranque automático quando são detectados proble-

mas de execução, o adaptador para recepção de notificações por socket permite precaver as plataformas cujos requisitos ao nível da utilização de memória sejam mais limitados e restrições ao nível arquitectural que impossibilitem a execução das APIs para comunicar com o intermediário.

- **Recepção de notificações**: O NGIN Manager usa o servidor aplicacional JBoss como ambiente de desenvolvimento da lógica de monitorização, desta forma a nova arquitectura requer a existência de Message Driven Beans para receber de forma assíncrona mensagens de monitorização por JMS, permitindo por um lado ter em conta a evolução dos componentes monitorizados tomando as medidas adequadas no sentido de precaver situações críticas e por outro lado receber mensagens com eventos que necessitem de ser tratados. A lógica de processamento das mensagens requer que o processo de gestão de inventário, desempenho e alarmes seja adaptado para que a partir de uma mensagem sejam registadas as entidades monitorizadas, sejam analisados as métricas de desempenho e extraídos possíveis eventos ocorridos.
  A desserialização de mensagens no NGIN Manager requer um descodificador diferente por protocolo de mensagens para a linguagem Java, desta forma é usado um protocolo de serialização de mensagens por defeito, no caso da existência de restrições arquitecturais este pode ser facilmente "alterado" por outro com a simples adequação da API de descodificação.
  Os componentes do NGIN Manager responsáveis por armazenar a informação numa data warehouse e de visualização da informação são independentes do processo de análise da informação pelo que não necessitam de ser alterados.

3.4 Componente de envio de eventos

A monitorização é efectuada ao nível da infra-estrutura, execução de serviços nas redes inteligentes e execução de operações dos componentes. Cada um destes níveis de monitorização engloba um conjunto extenso e heterogéneo de arquitecturas, aplicações ou plataformas, desta forma de modo a garantir a sistematização da estrutura de mensagens decidiu-se usar uma única estrutura por um lado, suficientemente genérica, para que possa ser usada em vários contextos e por outro lado flexível de forma a poder ser alterada sem quebrar a compatibilidade com versões anteriores do protocolo de mensagem. Por conseguinte, consegue-se ter uma única API, utilizada por componentes distintos, para serializar eventos em mensagens e uma única API de descodificação

simplificando o processo de desserialização de eventos no NGIN Manager.

3.4.1 Protocolo de serialização de mensagens

O uso de um protocolo de serialização de mensagens introduz facilidade na comunicação entre as diferentes entidades de uma rede, dado que abstrai o tipo de plataforma onde é serializada/desserializada a mensagem, permitindo a troca de informação entre todas as aplicações que suportem o protocolo usado. A desvantagem dos protocolos de serialização deve-se aos tempos de serialização, desserialização, de mapeamento da informação em objectos e ao overhead existente na informação serializada.
Os protocolos mais adequados às necessidades de interoperabilidade existentes foram referidos na secção 2.2.3, os resultados do benchmark apresentados na Tabela 2.3 indicam que de uma forma geral o Google Protocol Buffers é a opção mais indicada relativamente aos protocolos Apache Thrift e Apache Avro para estruturas de dados específicas e genéricas. No que concerne ao uso das normas ASN.1 e XDR, destaca-se a pouca flexibilidade do XDR, dado que o processo de desserialização impõe a obrigatoriedade de conhecer o tipo de dados. Por conseguinte, após comparar os protocolos ASN.1 e Google Protocol Buffers foi decido usar ASN.1, no entanto, este pode ser facilmente alterado por outro protocolo que faça mais sentido perante outros requisitos, bastando adaptar as API de serialização/desserialização como explicado na secção 3.4.3.
A decisão de usar ASN.1 deve-se ao facto de ser uma norma versada já implementada em diversas plataformas, por ter apresentado melhores resultados nos testes realizados e apresentados na Tabela 3.1 e por outro lado pelo facto de que a instalação de protocolos mais recentes como o Google Protocol Buffers implicarem a actualização demorada e complicada de ambientes de desenvolvimento e execução usados por toda a solução NGIN. O tipo de codificação ASN.1 usada será BER, dado que é uma solução que suporta codificação por CER e DER e cuja relação tempo de serialização/desserialização por tamanho da mensagem é mais equilibrada. As ferramentas disponibilizadas pela PT Inovação para serialização/desserialização utilizadas foram o *OSS Nokalva ASN.1* e *Objective Systems ASN.1 Compiler*.

Os testes realizados para comparar o Google Protocol Buffers com ASN.1 usaram o mesmo algoritmo, tipo de dados e valores do benchmark referido na secção 2.2.3. Os valores obtidos foram registados após várias execuções do procedimento de testes, cada teste é repetido 100 vezes e é composto por 2000 execuções sucessivas da mesma operação (serialização, desserialização ou criação de objecto), sendo o resultado final o mínimo da divisão do tempo obtido por 2000.

Tabela 3.1: Comparação dos protocolos de serialização Protocol Buffers e ASN.1

	Tempo de Serialização (nanosegundos)	Tempo de Desserialização (nanosegundos)	Tempo de Criação de Objectos (nanosegundos)	Tamanho da mensagem (bytes)
ASN.1	16338	33137	1493	229
Protocol Buffers	44744	31936	28343	245

Os resultados dos testes são apresentados na Tabela 3.1 e foram realizados numa máquina com as seguintes especificações:

- Sistema Operativo: Linux 2.4.21-20.EL
- Número de Processadores: 2
- Frequência de Relógio: 1396 MHz
- Memória RAM: 4110384 KBytes

3.4.2 Especificação da estrutura das mensagens

A especificação de estruturas de informação em ASN.1, permite o uso de tipos de dados como campos opcionais, escolhas, listas, entre outros, que introduzem flexibilidade ao ponto de criar uma estrutura suficientemente genérica em contextos diferentes. Uma notificação, como se pode ver na Figura 3.4 transporta informação relativa ao protocolo usado pelo componente monitorizado, ao contexto de execução da aplicação monitorizada, ao tipo de notificação, ao estado da execução de serviços do componente, à utilização de recursos de cpu e memória do componente na infra-estrutura onde se encontra instalado, à data de envio da última notificação, data actual e número identificador da notificação.
O contexto de execução da aplicação monitorizada é definido tendo em conta se é standalone, num ambiente controlado por aplicações externas (ControllerID) ou se é instalada em servidores aplicacionais (ServerID), como tal, uma notificação possui sempre dados da aplicação como por exemplo nome, versão e número de processo, como também, dados relativos ao ambiente onde são executados como o nome do cluster e nome do nodo num servidor aplicacional ou o nome e versão da aplicação que controla aplicações standalone. De forma a poder monitorizar qualquer aplicação, para além, dos dados habituais usados nestes contextos a estrutura da notificação possibilita adicionar dinamicamente novos campos e respectivos valores (Property). Uma notificação, de acordo com o cenário de execução, pode ser definida como sendo do tipo periódico,

final, a pedido ou de monitorização em tempo real.

O conteúdo da notificação com a definição exacta do estado do componente é definido através de métricas calculadas para um componente. A especificação ASN.1 define um componente como uma lista de secções e permite associar opcionalmente uma lista de métricas ao componente. Cada secção é representada através de uma lista dos vários tipos de métricas calculadas especificamente para a secção ou através de uma lista de serviços. Cada serviço é identificado por um nome e possui uma lista associada de métricas. O cálculo da utilização de CPU e memória é opcional dado que poderá não ser possível obter essa informação no ambiente de execução do componente monitorizado. As métricas calculadas podem conter simples valores associados a

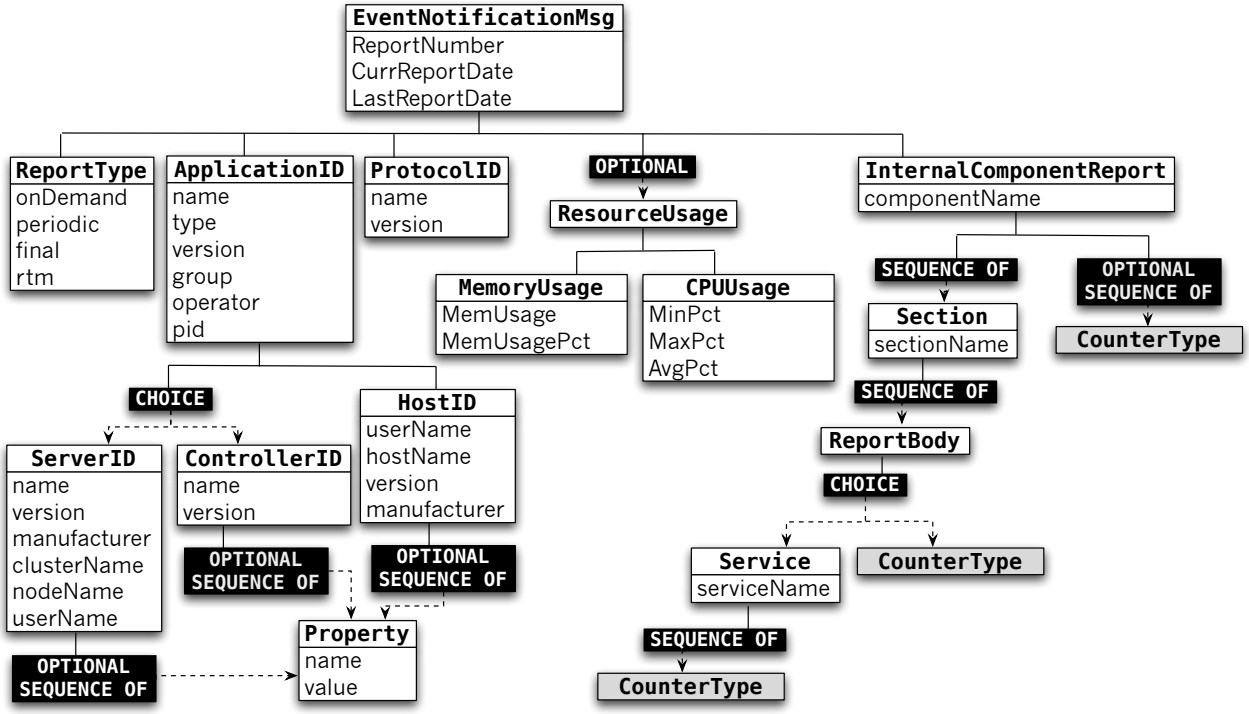

Figura 3.4: Estrutura de dados da notificação

entidades, ou índices com o ritmo de ocorrência de eventos, datas de execução de eventos, índices indicativos e índices com informação do sucesso, insucesso e descrição de erros ocorridos. Na Figura 3.5 são apresentados os diferentes tipos de dados para representação da informação de monitorização. Estas métricas foram especificadas de forma genérica, de modo a permitir a sua utilização em diferentes contextos.

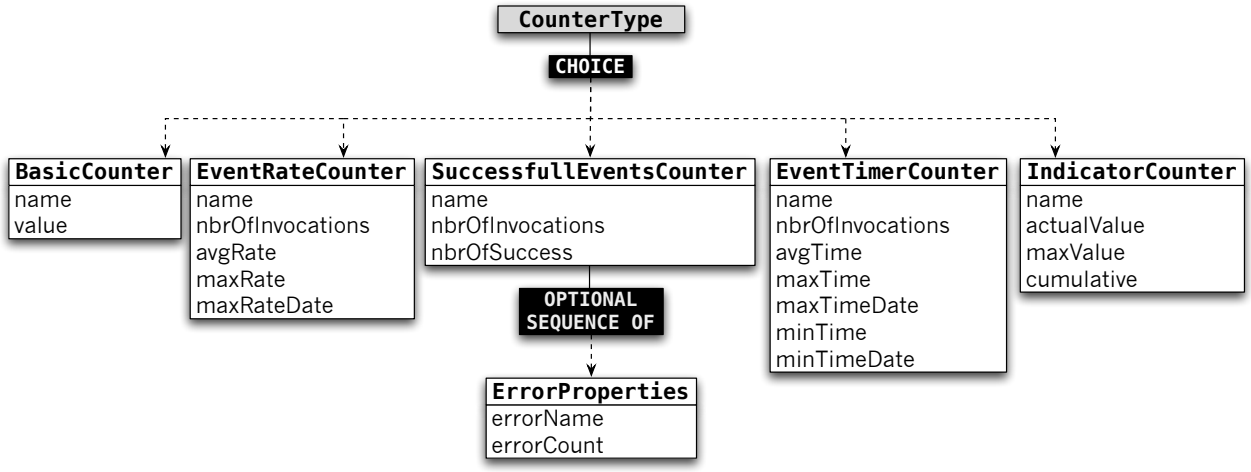

Figura 3.5: Tipos de métricas calculadas no componente monitorizado

3.4.3 Módulos de envio de eventos

Os módulos de envio de eventos integram directamente com os componentes monitorizados, pelo que, o processo de desenvolvimento depende do tipo da plataforma em que estes se encontram. Porém, dadas as funcionalidades de geração de código das ferramentas de serialização existentes e os diferentes protocolos de mensagens e transporte suportados pelo intermediário, o processo de desenvolvimento de novos módulos é simplificado. De forma a demonstrar o processo de desenvolvimento de novos módulos foi implementado para a plataforma C/C++ uma biblioteca de serialização de mensagens em ASN.1 e envio através do protocolo OpenWire ou sockets TCP. A decisão de suportar o envio de notificações também por sockets, deve-se à menor ocupação de memória comparativamente ao envio usando o protocolo OpenWire, indo de encontro aos requisitos da solução. O módulo desenvolvido permite a integração com a lógica de monitorização de qualquer componente da plataforma C/C++, tendo sido utilizado o componente SCF da rede inteligente para esse efeito.

A geração de código foi efectuada utilizando a ferramenta *OSS Nokalva ASN.1 para C* disponibilizada pela PT Inovação, o código gerado consiste num conjunto de structs para cada tipo de dados estruturado do ASN.1, listas ligadas para as sequências usadas, unions para as diferentes alternativas existentes e bitmasks para identificar a presença dos atributos opcionais. O diagrama com as estruturas criadas é apresentado na Figura 3.6, recorrendo aos modelos UML de diagramas de classes.

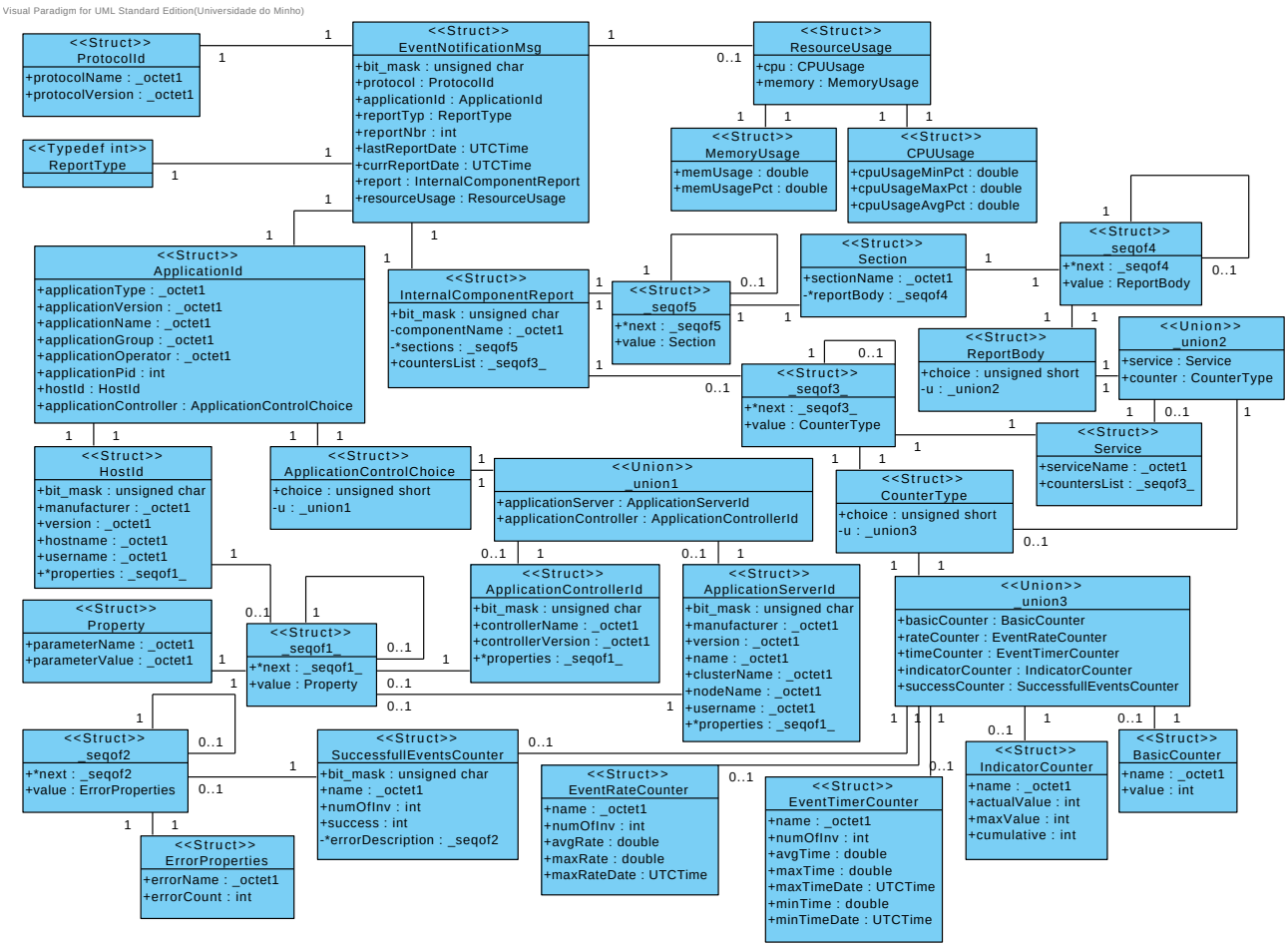

Figura 3.6: Diagrama do código gerado em C

A implementação da biblioteca de serialização e envio de eventos foi realizada tendo em conta as necessidades existentes no consumo de memória das máquinas monitorizadas, desta forma, o envio de eventos pode ser realizado utilizando um socket ou através do protocolo OpenWire suportado pelo intermediário. O intermediário de eventos e a forma de envio é definido num ficheiro de configuração, permitindo a sua redefinição em tempo de execução. De forma, a suportar eventuais indisponibilidades no intermediário de eventos, existe um buffer em memória, cujo limite é definido por configuração, e que armazena os eventos que não tenham sido enviados, até que o intermediário fique disponível ou até ser atingido o limite máximo do buffer. Na Figura 3.7 é apresentado um extracto do diagrama de classes do desenvolvimento deste módulo, onde se destaca a especificação de uma thread de cálculo das percentagens de memória e cpu utilizadas pelos componentes monitorizados e a integração com o nodo principal da estrutura de dados do código gerado, isto é, a estrutura EventNotificationMsg. Dado que a

classe EventNotificationCls possui uma API de definição da estrutura de informação demasiado extensa, não foram apresentadas as funções e classes associadas, no entanto resumem-se aos métodos utilizados pelos componentes monitorizados para definir a informação em ASN.1. A

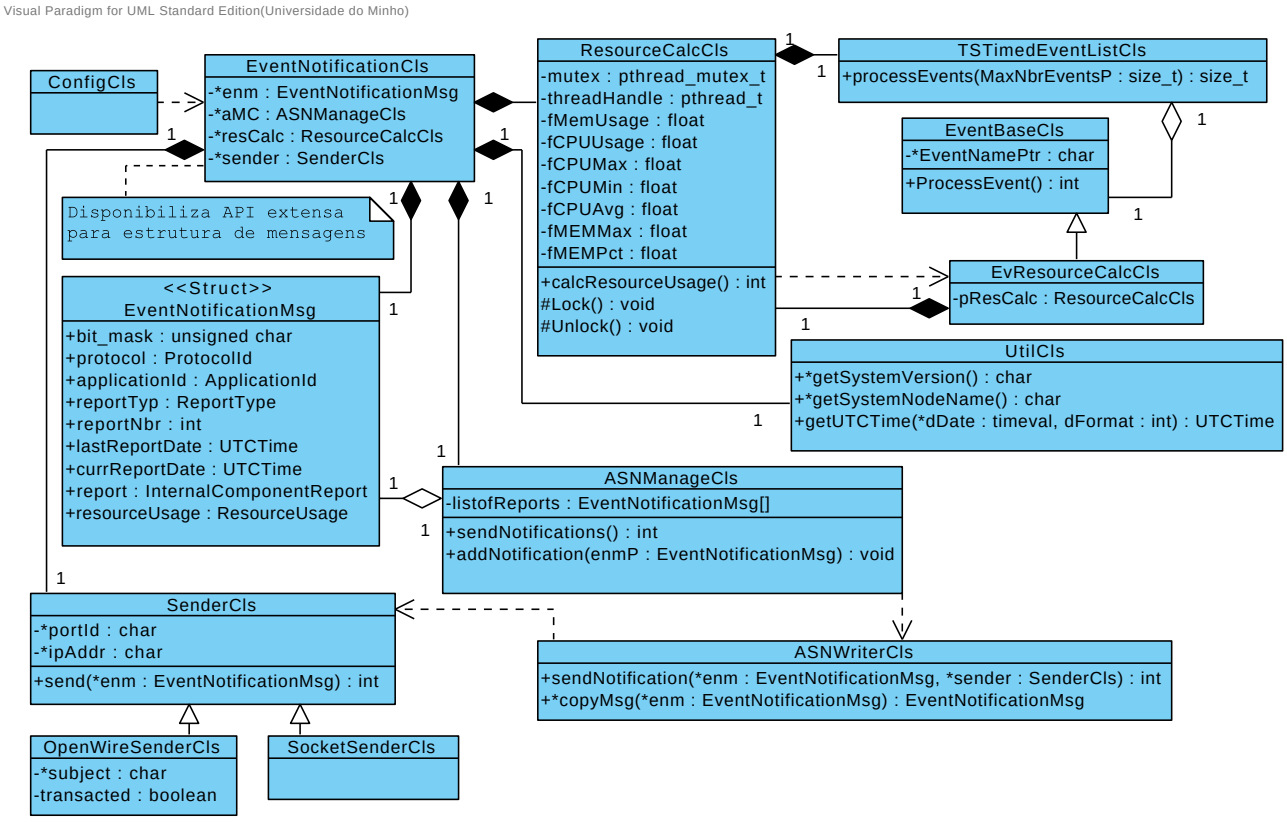

Figura 3.7: Biblioteca C/C++ para serialização e envio de notificações

integração do módulo desenvolvido pode ser integrado com todos os componentes cujo ambiente de execução seja a plataforma C/C++, para demonstrar o seu funcionamento foi integrado com o componente SCF da rede inteligente. Desta forma foi necessário adequar as bibliotecas de recolha de informação para utilizarem a nova API e instalar o novo binário num ambiente onde se possa efectuar a lógica de controlo de sinalização.

3.5 Componente de intermediação de eventos

O intermediário de mensagens assume um papel fulcral na nova arquitectura, pois tem a responsabilidade de introduzir fiabilidade na comunicação entre os componentes de monitorização e o NGIN Manager para processamento da informação. Por outro lado, é essencial que o in-

termediário de mensagens permita a comunicação e integração entre plataformas heterogéneas através da disponibilização de diferentes protocolos de mensagens e transporte. No capítulo 2 foram apresentadas diferentes alternativas para a definição de arquitecturas orientadas a eventos, com garantias de fiabilidade, segurança e que permitam a fácil integração de novos componentes. Os ESBs surgem como uma alternativa demasiado específica, incluindo um conjunto vasto de tecnologias que disponibilizam serviços de orquestração, transformação, encaminhamento, integração, intermediação, entre outros, o que é mais do que necessário para a definição da arquitectura. Desta forma, foi decidido usar uma solução modular, apenas com os componentes necessários e que possa evoluir integrando novas tecnologias e soluções sem haver uma imposição tecnológica, como tal, a solução actual implica a utilização uma plataforma de mensagens que garanta a fiabilidade na comunicação e que suporte diferentes protocolos de mensagens de forma a poder ser integrado em plataformas heterogéneas e a utilização de uma plataforma que introduza uma lógica de encaminhamento de mensagens à arquitectura.
Na secção 2.1.2 são apresentadas as plataformas de WS-*, ORBs e MOMs como alternativas para plataformas de mensagens, apesar de todas possuírem especificações para garantir a fiabilidade, segurança, suporte para transacções e comunicação assíncrona as MOMs apresentam-se como a alternativa mais adequada. As soluções baseadas em ORBs têm tipicamente problemas de escalabilidade e de latência quando usados em ambientes de alta performance. As soluções baseadas em Web Services, apesar de possuírem especificações WS-* que asseguram requisitos de alta disponibilidade, usam, em alguns casos, MOMs para garantir comunicações assíncronas e fiáveis, como por exemplo ESBs que integram MOMs com Web Services e que usam SOAP com JMS. Além disso, existe pouca performance dos Web Services com especificações de fiabilidade em HTTP comparativamente aos mecanismos existentes de persistência e caching das MOMs.
Deste modo, comparando as funcionalidades dos intermediários apresentados na secção 2.1.5, destaca-se o Apache ActiveMQ pelas capacidades de integração, dada a diversidade de protocolos, linguagens suportadas e por ser disponibilizado com a solução Apache Camel que disponibiliza mecanismos de encaminhamento e padrões de integração.

3.5.1 Descrição das funcionalidades

O intermediário de mensagens realiza funções de integração, intermediação e encaminhamento através das plataformas Apache ActiveMQ e Apache Camel. O ambiente de desenvolvimento disponibilizado por estas soluções é Java e baseia-se no uso da plataforma Spring Framework que simplifica o processo de configuração das funcionalidades do intermediário, no entanto, todas as

funcionalidades podem ser especificadas usando directamente as APIs existentes. As principais funcionalidades disponibilizadas pelo intermediário de mensagens são:

- Persistência: A persistência pode ser realizada numa simples base de dados JDBC ou através de mecanismos de persistência disponibilizados pelas plataformas *AMQ Message Store* e *Apache KahaDB*. O AMQ Message Store permite persistir rapidamente notificações, enquanto que a KahaDB, que vem integrada com o ActiveMQ, possui melhorias de escalabilidade e recuperação da informação. O processo de persistência usa também mecanismos de journaling e "Message Cursors" de forma a aumentar a eficiência. Na implementação do intermediário foi usada a KahaDB para persistência de informação em conjunto com os mecanismos eficientes do ActiveMQ.

- Escalabilidade: O intermediário oferece funcionalidades que permitem aumentar a escalabilidade, tal como, a criação de uma rede de intermediários com protocolos de failover, fanout, multicast, etc.

- Protocolos de Mensagens e Linguagens suportadas: A utilização do intermediário por parte de clientes em plataformas heterogéneas depende essencialmente dos protocolos de mensagens suportados e consequentemente das linguagens suportadas por cada protocolo de mensagens. As principais linguagens e respectivos protocolos suportados são Java (JMS), C/C++ (Stomp, OpenWire), Ajax (Rest), Python (Stomp, OpenWire), C# e .NET (NMS API), entre outros.

- Controlo de fluxo: O fluxo de mensagens recebidas pode ser controlado pelo intermediário se as mensagens enviadas forem síncronas ou pelo produtor de mensagens no caso de serem assíncronas. O intermediário faz o controlo de fluxo através de um limite no tamanho máximo das mensagens recebidas, limites de memória, limites de tamanho em disco e limites de mensagens temporárias. Se o limite for atingido o intermediário deixa de enviar mensagens ack, abrandando o ritmo de mensagens enviadas. No caso do envio assíncrono de mensagens o produtor de acordo com a disponibilidade do sistema deve definir o número máximo de bytes a enviar antes de ficar à espera de mensagens de confirmação (ack).

- Controlo e monitorização do intermediário: O intermediário pode ser configurado para dar permissões de monitorização e controlo por JMX de forma a que se possa monitorizar fluxos de conexões, sessões, produtores, consumidores, destinos, controlo de queues e tópicos e do próprio intermediário, podendo ser parado remotamente. Para além de JMX, foram implementadas funcionalidades no intermediário para que este possa ser controlado

por uma ferramenta existente na PT Inovação, permitindo arrancar e parar o intermediário remotamente e detectando eventuais fins inesperados da aplicação procedendo ao arranque automático.

- Segurança: O intermediário pode ser configurado para ter diferentes políticas de autenticação e autorização, usando o serviço JAAS (Java Authentication and Authorization Service) que pode ser integrado com o LDAP ou um mecanismo próprio do intermediário baseado em configurações XML.

- Encaminhamento e suporte transaccional: O intermediário possui um conjunto vasto de padrões de integração empresariais (EIP) que podem ser implementados usando a plataforma Apache Camel. O encaminhamento entre plataformas de mensagens é indispensável para que os eventos que chegam a determinadas queues do intermediário possam ser entregues ao NGIN Manager. O suporte transaccional do Apache Camel é indispensável para garantir sempre a entrega das notificações independentemente do estado do NGIN Manager.

- Adaptador de recepção de mensagens por socket: A recepção de mensagens do intermediário pode ser configurada para comunicar via sockets TCP/IP, tendo sido implementado um adaptador que, para cada mensagem recebida, cria uma BytesMessage JMS e a insere no destino apropriado.

3.5.2 Adaptador de recepção de eventos

Muitos dos componentes monitorizados encontram-se presentes em máquinas cuja utilização de recursos de memória é limitada, desta forma, para precaver estas situações foi implementado um adaptador no intermediário de mensagens que permite a recepção de notificações através de um socket TCP/IP. Esta situação permite minimizar o impacto das APIs existentes para utilização dos protocolos de mensagem suportados pelo intermediário, permitindo inclusive precaver a monitorização de componentes cuja arquitectura possa não estar preparada para usar um dos protocolos existentes. O adaptador encontra-se activo numa porta parametrizada por configuração, após a recepção de uma notificação, esta é inserida numa queue especificada por configuração. Como apresentado na Figura 3.8, a implementação do adaptador envolve a leitura de um ficheiro de configuração XML através da classe XMLConfiguration, esta classe instância um map com as portas em que os servidores socket vão ficar à espera de mensagens como chave e o destino das

notificações como valor. Quando um servidor recebe uma notificação, é criada uma BytesMessage e colocada num buffer da classe RequestBuffer. A responsabilidade de persistir a notificação no intermediário é da thread executada na classe AMQEventProducer que previamente retira a notificação da classe RequestBuffer.

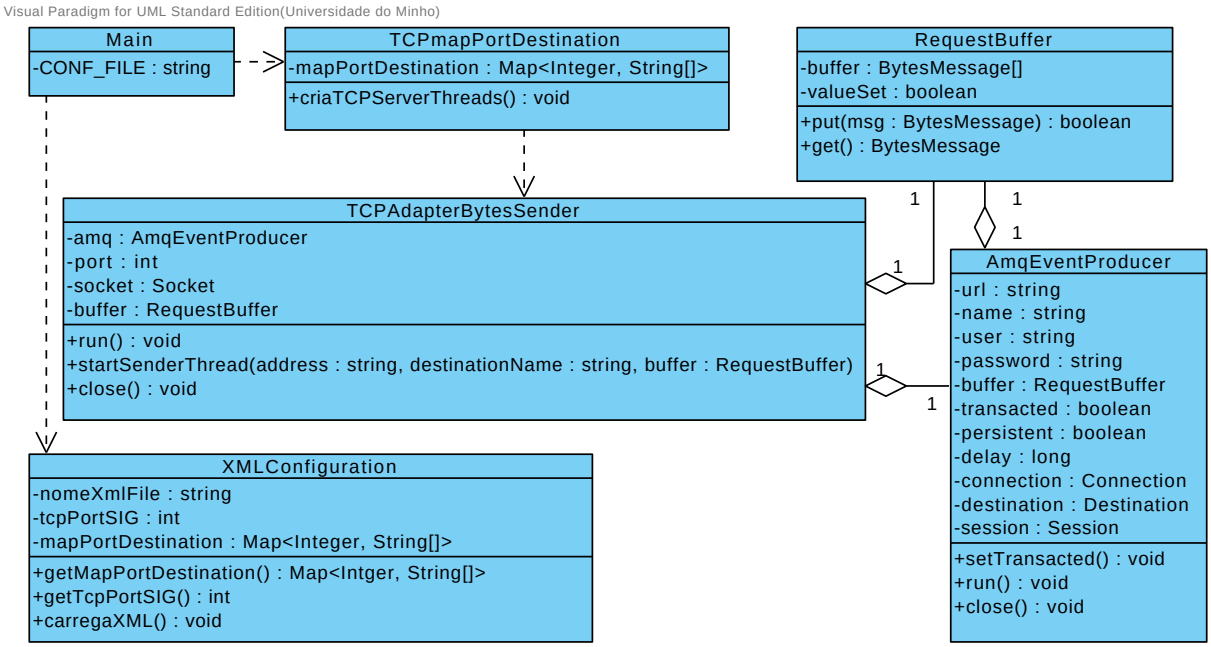

Figura 3.8: Diagrama de classes do adaptador de recepção de eventos

3.5.3 Módulo de controlo do intermediário

O módulo de controlo do intermediário de mensagens tem como principal funcionalidade garantir o correcto funcionamento do intermediário, permitindo arrancar ou parar remotamente o intermediário e detectando eventuais paragens inesperadas da aplicação, procedendo imediatamente ao seu arranque. A PT Inovação possui a ferramenta LSC (Local Subsytem Controller) que funciona com aplicações que recebem sinais nativos do sistema operativo, desta forma para integrar com o intermediário que é executado numa JVM foi desenvolvido um módulo em Perl que recebe os sinais do LSC e que envia mensagens equivalentes através de um socket. O módulo implementado no intermediário possui um socket activo à espera da recepção de informação que indique a ocorrência desses sinais. Na Figura 3.9 é apresentado o diagrama de classes da implementação que trata os sinais enviados pelo LSC ao módulo desenvolvido em Perl. Existe uma interface que define os métodos que devem ser implementados para tratar os sinais, esta in-

terface é implementada pela classe Main, dado o maior controlo que tem sobre as outras classes. Os sinais são recebidos através de um socket executado numa thread da classe SIGRecvServer e cuja porta é lida de um ficheiro de configuração através da classe XMLConfiguration, por cada sinal recebido é criada uma nova thread com a classe SIGRecvThread que verifica o tipo de sinal e invoca o método respectivo da interface.

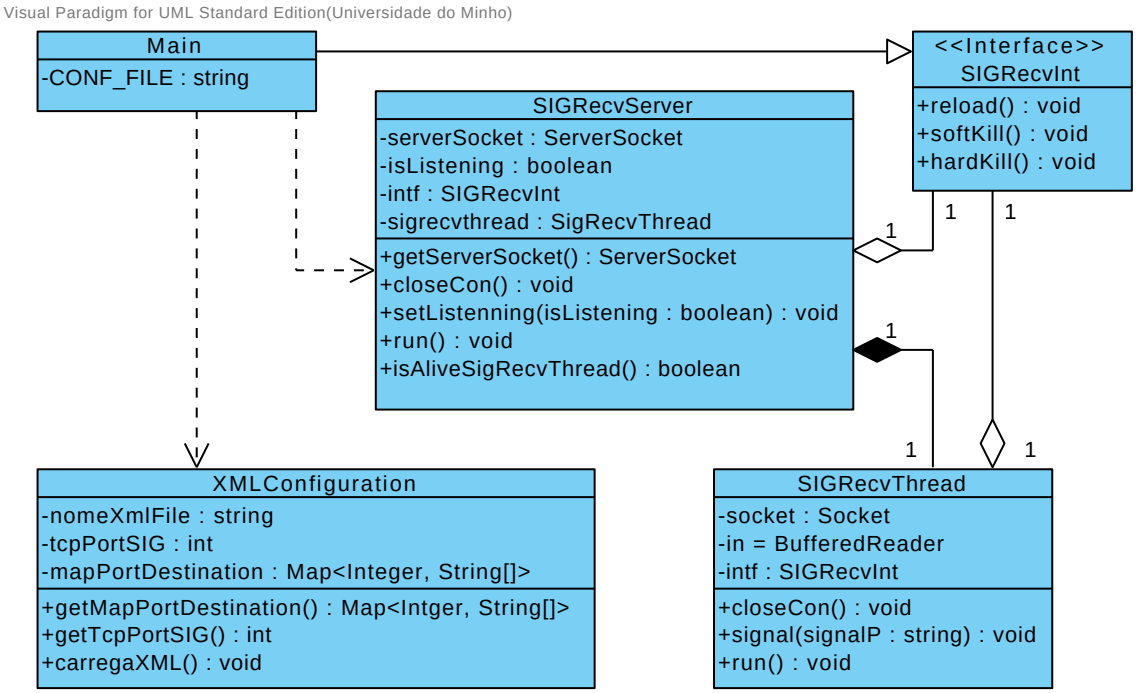

Figura 3.9: Diagrama de classes do controlador do intermediário

3.5.4 Encaminhamento de mensagens

O intermediário tem como principal função garantir a fiabilidade na comunicação entre os componentes monitorizados e a plataforma NGIN Manager, para tal usa mecanismos de journaling e persistência de alto desempenho, tentando fazer o encaminhamento das mensagens o mais rápido possível. O encaminhamento é implementado com a plataforma Apache Camel que para todas as notificações recebidas numa determinada queue, define qual o destino JMS que as irá receber na plataforma JBoss instalada no NGIN Manager. De forma a contemplar situações em que a a plataforma NGIN Manager possa estar indisponível o encaminhamento é configurado com transacções, permitindo manter a notificação persistida no intermediário com um rollback da operação de envio. No momento em que a plataforma NGIN Manager fica disponível são

imediatamente enviadas todas as notificações persistidas até ao momento.
O encaminhamento das notificações introduz uma maior flexibilidade na gestão da lógica de monitorização dado que permite a rápida configuração de novas regras, isto é, a monitorização de diferentes componentes pode ser dividida em diferentes queues no intermediário ao qual está associada uma regra de encaminhamento diferente, além disso, permite gerir notificações serializadas com protocolos distintos, simplificando a lógica de negócio do NGIN Manager.

3.6 Componente de análise de eventos

O processamento das notificações é efectuado no back-end do NGIN Manager com o servidor aplicacional JBoss. A recepção das notificações é efectuada através de uma queue à qual se encontra associado um Message Driven Bean (MDB), de cada vez que uma notificação JMS chega à queue o MDB vai invocar a lógica de processamento da informação. Desta forma, pode ser configurado um MDB por componente monitorizado ou por protocolo de serialização, invocando para cada notificação o processo de descodificação e armazenamento da informação numa data warehouse. Posteriormente existe um pós processamento da informação por parte da gestão de desempenho e gestão de alarmes do NGIN Manager.

3.6.1 Integração com o servidor aplicacional do NGIN Manager

A integração com a plataforma NGIN Manager requereu a implementação de um descodificador de mensagens ASN.1, como tal, foi usada a ferramenta *Objective Systems ASN.1 Compiler* para gerar código fonte em Java a partir da especificação ASN.1.

Na Figura 3.10 é apresentado um excerto do código gerado para Java a partir da especificação ASN.1 usado para desenvolver um MBean no JBoss que disponibilize um serviço de descodificação. Dependendo da ferramenta usada o código pode ser mais fácil de usar, tal como se pode ver nas classes geradas onde existem métodos e variáveis específicas para criar e alterar a estrutura ASN.1.

O processamento da notificação recebida, passa pela descodificação e invocação da lógica de armazenamento da informação na data warehouse. Essa informação é obtida pela solução front-end do NGIN Manager que disponibiliza em interfaces Web informação de monitorização, onde podem ser geridos alarmes entre outras funcionalidades.
Desta forma, a integração do NGIN Manager com a nova arquitectura de monitorização foi

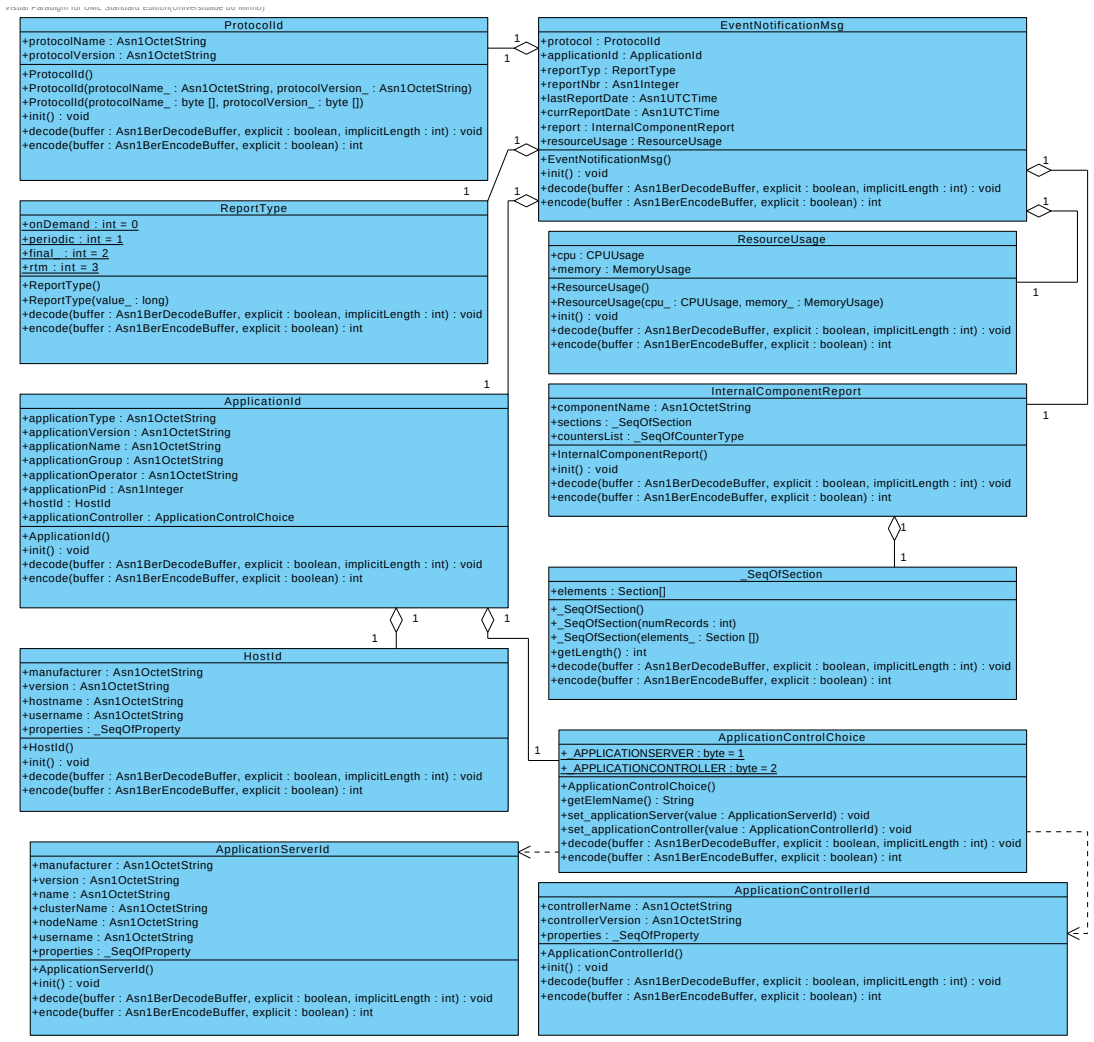

Figura 3.10: Diagrama de classes com excerto do código gerado para Java

realizada ao nível do back-end através dos MBeans de processamento e de descodificação da notificação.

3.7 Sumário

Neste capítulo foi apresentado o trabalho envolvido na análise, especificação e desenvolvimento da arquitectura orientada a eventos. Inicialmente foi apresentada a situação actual do NGIN Manager, sendo descritos os componentes monitorizados pela solução actual e as principais necessidades de alteração nos mecanismos de monitorização, destacando em especial a garantia de fiabilidade como uma funcionalidade imprescindível na comunicação entre os agentes de mo-

nitorização e o NGIN Manager. Posteriormente, são enunciados os requisitos da nova solução e é apresentada a arquitectura da solução dividida nos componentes de envio, intermediação e recepção de notificações. Para o componente de envio de notificações é apresentado o desenvolvimento realizado, as decisões tomadas para o uso de protocolos de serialização ainda que possam ser facilmente alteráveis e é descrita a estrutura de uma notificação que será construída pelo agente de monitorização. Para o componente de intermediação de eventos são apresentadas as funcionalidades da tecnologia escolhida, o desenvolvimento de módulos no intermediário para possibilitar a recepção de notificações também por socket, o desenvolvimento do módulo de controlo da aplicação e descrito o processo de encaminhamento das notificações. O componente de recepção e análise de notificações é apresentado descrevendo a forma como são recebidas e processadas as notificações.

Capítulo 4

Cenários de utilização e Testes

Neste capítulo são apresentados diferentes cenários de utilização da solução e é feita uma análise dos resultados obtidos para os diferentes testes desenvolvidos. Na secção 4.1 são apresentados detalhadamente os vários contextos de execução implementados de forma a garantir a fiabilidade da arquitectura. Na secção 4.2 são apresentados os resultados dos testes que visam analisar o desempenho da solução em diferentes contextos. A execução dos testes foi realizada num ambiente simulado que permite simular o funcionamento real do componente SCF da solução NGIN.

4.1 Cenários de utilização

A arquitectura de monitorização criada foi desenhada e implementada de forma a disponibilizar mecanismos de monitorização que garantam os requisitos exigentes das operadoras de telecomunicações, possibilitando a sua adequação a diferentes ambientes de execução dos componentes da rede inteligente e de subsistemas da solução NGIN. Dado que o contexto de monitorização se enquadra num ambiente com operações solicitadas em tempo real por vários clientes, a fiabilidade na comunicação entre os diversos componentes de monitorização da solução tem um papel fundamental, garantindo que a informação de monitorização é sempre processada. Neste contexto, os cenários de utilização visam demonstrar o comportamento da arquitectura perante casos de indisponibilidade na comunicação entre os componentes monitorizados e a plataforma NGIN Manager, responsável pelo pós-processamento da informação.

4.1.1 Funcionamento normal da arquitectura

O funcionamento normal da arquitectura implica que todos os componentes estejam operacionais, isto é, a recolha de informação do componente monitorizado e consequente envio, a recepção, armazenamento e encaminhamento de notificações do intermediário de mensagens e a recepção bem sucedida por parte do NGIN Manager. Este cenário é apresentado no diagrama de sequência da Figura 4.1, onde são descritas as operações efectuadas para cada entidade da arquitectura agrupando conjuntos de operações em cortes horizontais a), b) e c).

No corte horizontal a) o agente instalado no ambiente de execução do componente monitorizado recolhe estatísticas de execução dos serviços e de toda a infra-estrutura em si, essa informação é serializada em ASN.1 ou através de outro protocolo de serialização e antes de ser enviada é previamente armazenada num buffer em memória, este conjunto de operações é representado na Figura 4.1 no corte horizontal a). Este buffer possui um tamanho definido por configuração e permite armazenar notificações que não tenham sido enviadas devido à indisponibilidade do intermediário. Uma vez que o intermediário de mensagens pode ser facilmente configurado para funcionar em modo cluster com sistemas de failover e dado que foram implementadas funcionalidades no intermediário que possibilitam o seu controlo pela aplicação LSC (Local Subsytem Controller) o tempo de indisponibilidade no intermediário será muito reduzido, evitando que o buffer de envio tenha que suportar demasiadas mensagens.

No corte horizontal b) é descrito o processo de recepção de notificações pelo intermediário, que para cada notificação recebida efectua de imediato a persistência da notificação seguindo-se o envio da confirmação de recepção ao agente e o encaminhamento para o Message Driven Bean do NGIN Manager. O agente de envio após receber a confirmação enviada pelo intermediário elimina a notificação do buffer. O encaminhamento para o Message Driven Bean foi implementado com uso de transacções permitindo efectuar rollbacks das operações de envio em caso de indisponibilidade do NGIN Manager.

Por fim, como apresentado no corte horizontal c) o NGIN Manager recebe as notificações através de um Message Driven Bean que após confirmar a recepção da notificação invoca os serviços de descodificação e processamento da notificação. O serviço de descodificação instância um objecto Java a partir de um conjunto de bytes recebidos numa mensagem JMS (BytesMessage). O serviço de processamento armazena a informação de monitorização numa data warehouse em diferentes dimensões de acordo com o tipo de entidade monitorizada e de métrica calculada.

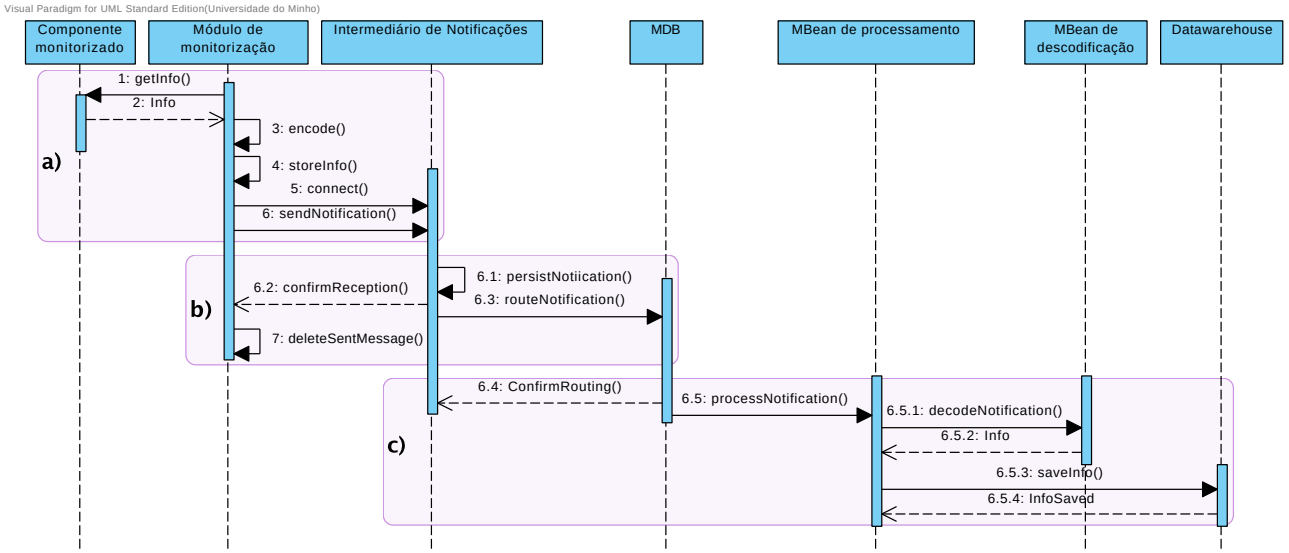

Figura 4.1: Cenário de funcionamento normal da arquitectura

4.1.2 Cenário de indisponibilidade do NGIN Manager

A indisponibilidade do NGIN Manager enquadra-se no conjunto de situações que a arquitectura suporta. A indisponibilidade poderá ocorrer essencialmente perante situações em que existem problemas de rede, no entanto, independentemente da causa de indisponibilidade, o intermediário possui configurada uma base de dados onde são persistidas todas as notificações recebidas. Uma notificação só é removida da base de dados quando o intermediário conseguir realizar uma conexão e receber a confirmação de recepção da notificação por parte do MDB configurado no servidor aplicacional do NGIN Manager. De forma a minimizar a perda de desempenho com o processo de "store and forward" das notificações, o intermediário usa sistemas de cache e journaling que permite garantir a persistência das notificações antes de proceder ao seu encaminhamento com alto desempenho.

Na Figura 4.2 é apresentada o diagrama de sequência num cenário de indisponibilidade do NGIN Manager, neste cenário o agente de envio de notificações possui exactamente o mesmo comportamento do cenário de funcionamento normal, isto é, a informação de monitorização recolhida no componente monitorizado e da infra-estrutura é serializada em ASN.1, armazenada num buffer em memória e posteriormente enviada. Neste cenário o intermediário após receber a notificação e de a persistir numa base de dados não vai conseguir estabelecer uma ligação com o NGIN Manager, como tal, o intermediário não vai eliminar a notificação persistida, tentando enviar periodicamente todas as notificações recebidas ao Message Driven Bean do NGIN Manager.

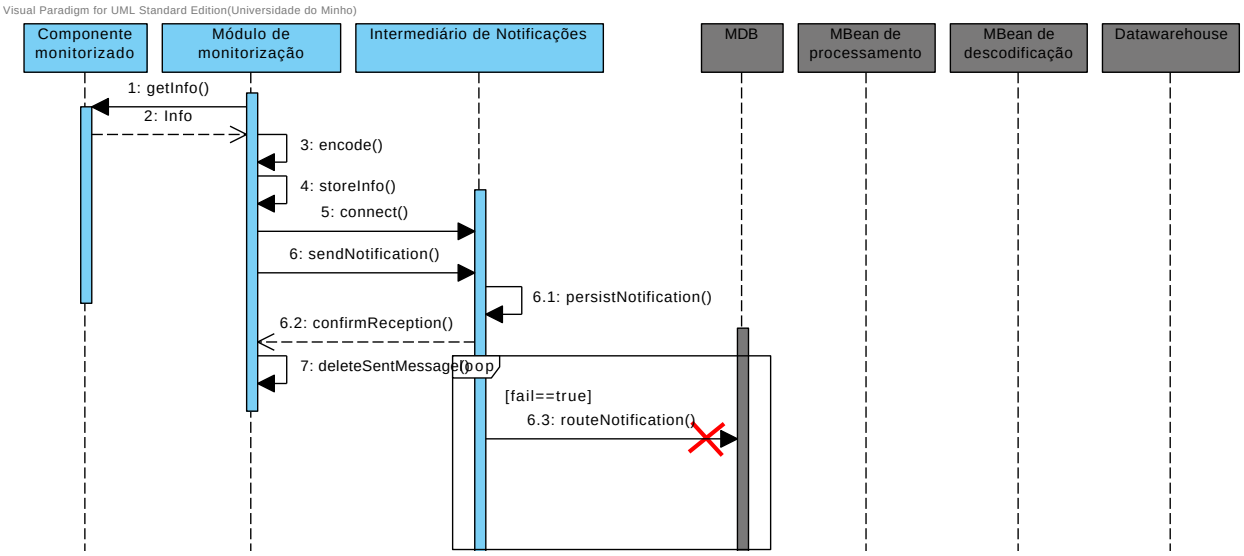

Figura 4.2: Cenário de indisponibilidade do NGIN Manager

4.1.3 Cenário de indisponibilidade do intermediário de notificações

A indisponibilidade do intermediário de notificações é um cenário que poderá acontecer, no entanto apenas durante um curto intervalo de tempo. Caso se tenha configurado o intermediário como um elemento de um cluster a notificação irá ser recebida por outro intermediário, caso exista apenas um intermediário este será automaticamente reiniciado dado que foi implementado no intermediário a funcionalidade de interacção com a ferramenta LSC da PT Inovação. No caso de se usar a ferramenta LSC, o intermediário será automaticamente reiniciado, como tal, o número de notificações que necessitam de ser mantidas em memória pelo agente de envio de notificações depende do tempo que o intermediário demora até ficar novamente operacional. Nesta situação, conforme apresentado na Figura 4.3, o agente de envio após recolher a informação de monitorização de um componente e da infra-estrutura procede à serialização em ASN.1 e armazenamento no buffer em memória, falhando no respectivo envio ao intermediário. Como tal, a notificação não será eliminada do buffer até que se consiga enviar todas as mensagens existentes no buffer ou até que o este fique sem capacidade para novas notificações, dado que é dada maior prioridade a novas notificações.

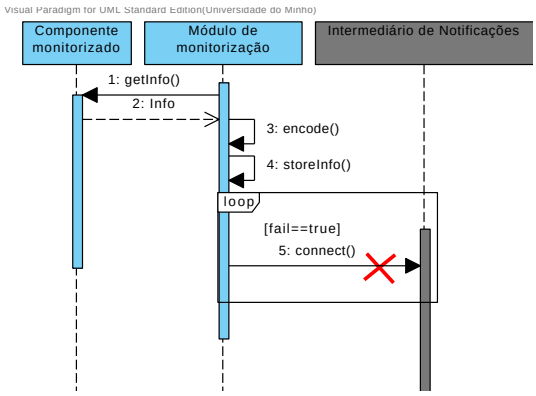

Figura 4.3: Cenário de indisponibilidade do intermediário

4.2 Testes realizados

De seguida são apresentados os testes e respectivos resultados obtidos realizados tendo em conta os cenários apresentados na secção 4.1, pretendendo-se desta forma avaliar o desempenho da solução nos contextos que asseguram a fiabilidade de comunicação. Como é de esperar, em cenários cujo objectivo é sobretudo garantir a fiabilidade, os resultados de desempenho não serão tão bons como em situações onde a perda de notificações não tem qualquer impacto para o negócio. O principal componente responsável por garantir a fiabilidade é o intermediário de eventos, que só encaminha as notificações recebidas após as persistir numa base de dados, no entanto, utilizando mecanismos de cache e journaling consegue-se aumentar o desempenho para valores aceitáveis. Para além do intermediário o agente de envio de notificações usa um buffer em memória onde são guardadas as notificações que não foram enviadas. Dadas as restrições de utilização de memória nas máquinas monitorizadas o tamanho máximo do buffer pode ser indicado através de ficheiros de configuração. Para a correcta definição do limite do buffer de notificações no agente dever-se-á ter em conta o tempo de arranque do intermediário de eventos e a frequência com que serão enviadas as notificações.

Na execução destes testes, as notificações enviadas possuem todas aproximadamente 4170 bytes e são usadas três máquinas distintas, uma onde é simulada a execução de um SCF e onde se encontra o agente que o monitoriza enviando notificações para o intermediário de eventos, na segunda máquina está configurado o intermediário que persiste as notificações recebidas e as encaminha para o NGIN Manager e por fim a terceira máquina onde está a ser executado o NGIN Manager com um MDB associado a uma queue de eventos e onde se encontram os serviços de processamento e descodificação da notificação. As características das máquinas são apresentadas na Tabela 4.1.

Tabela 4.1: Descrição das características das máquinas onde foram realizados os testes

	Envio	Intermediação	Recepção
Sistema Operativo	HP-UX B.11.11	Linux 2.4.21-20.EL	Linux 2.6.9-55.ELsmp
Número de processadores	2	2	4
Frequência de Relógio	1000 MHz	1396 MHz	3061 MHz
Memória RAM	1756476 KBytes	4110384 KBytes	3115200 KBytes

4.2.1 Testes de carga/fiabilidade da solução

Os resultados dos testes de desempenho num cenário de funcionamento normal são apresentados na Tabela 4.2 e no gráfico da Figura 4.4. Na tabela são apresentados para um conjunto de notificações enviadas sequencialmente, a memória utilizada na serialização e envio, o tempo de serialização e envio de notificações, o tempo de recepção das notificações no intermediário e o tempo que o MDB no NGIN Manager demora a receber as notificações reencaminhadas pelo intermediário.

Tabela 4.2: Resultados dos testes de desempenho num cenário de funcionamento normal

Número de Notificações Enviadas	Memória utilizada (KBytes)	Tempo de serialização e envio (mi:ss.ms)	Tempo de Recepção do Intermediário (mi:ss.ms)	Tempo de Recepção do MDB (mi:ss.ms)
100	2896	01:48.722	01:48.492	01:43.943
200	2896	03:30.805	03:30.580	03:26.055
300	2896	05:19.041	05:18.838	05:14.415
400	2896	07:03.217	07:02.858	06:58.467
500	2888	08:59.417	08:59.239	08:55.212
600	2896	10:34.722	10:34.556	10:29.959
700	2896	12:34.614	12:34.705	12:30.281
800	2896	15:41.841	15:41.734	15:37.228
900	2768	16:08.631	16:08.597	16:03.826

Os resultados da memória utilizada na serialização e envio permitem concluir que apesar do aumento do número de notificações a memória utilizada mantém-se estável, este facto deve-se essencialmente à gestão cuidada da memória realizada pelas aplicações, permitindo-se desta forma que as aplicações possam ter períodos de execução ilimitados. Os tempos de serialização e envio de notificações permitem concluir que em média é necessário um segundo para serializar uma notificação em ASN.1 e proceder ao seu envio. Os tempos de recepção de notificações no

intermediário são menores que os tempos de serialização e envio permitindo constatar o peso da serialização antes do envio e por outro lado permite verificar que o intermediário se adequa bem a situações de maior fluxo. Os tempos de recepção do MDB configurado no servidor aplicacional JBoss do NGIN Manager são menores que os tempos de recepção no intermediário. Este facto deve-se aos atrasos existentes no estabelecimento das conexões e sessões para a primeira notificação, pois o intermediário após receber cada notificação procede à persistência em base de dados e coloca cada notificação recebida numa queue lógica, posteriormente é feito o encaminhamento das notificações existentes na queue local para outra queue configurada no JBoss do NGIN Manager. Apenas neste instante é que o MDB vai ter acesso à primeira notificação, entretanto as restantes notificações que foram recebidas e persistidas pelo intermediário são encaminhadas na mesma sessão já estabelecida com o NGIN Manager.

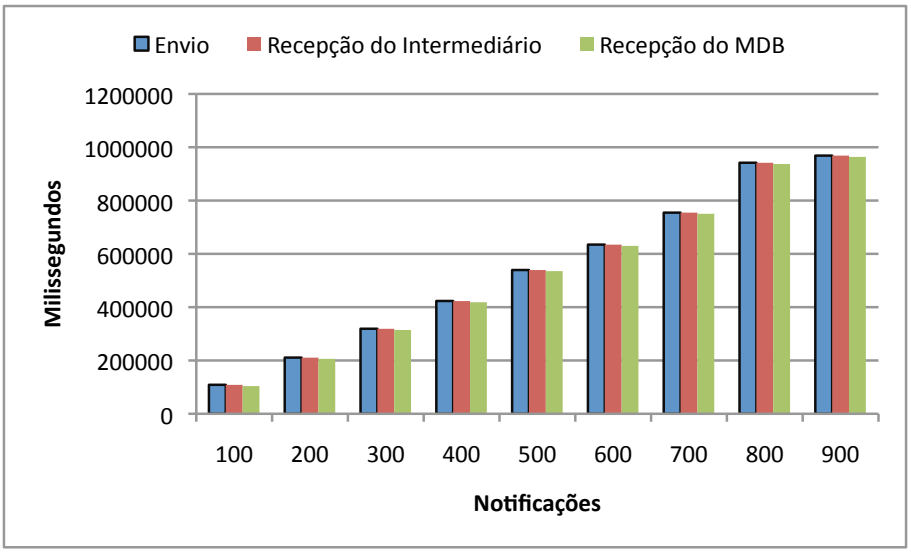

Figura 4.4: Resultados dos testes de desempenho num cenário de funcionamento normal

O intermediário de notificações é o ponto central da arquitectura que oferece garantias de fiabilidade a toda a solução, no entanto de forma a ter em conta situações em que o próprio intermediário fique indisponível, foram implementadas funcionalidades que permitem minimizar ou mesmo solucionar este problema. Desta forma se o intermediário estiver configurado num cluster com vários intermediários e ficar indisponível, as novas notificações serão automaticamente enviadas para outro intermediário do cluster. Dado que este mecanismo pode representar ter demasiadas máquinas, cada uma activa com um intermediário foi desenvolvida uma solução que permite que um intermediário interaja com a ferramenta LSC da PT Inovação. Esta ferramenta envia sinais às aplicações para as parar ou arrancar, detecta quando a aplicação termina

reiniciando-a automaticamente e pode ser administrada remotamente. Desta forma, caso a aplicação termine inesperadamente a ferramenta de controlo irá proceder ao seu reinicio, no entanto, durante o período de arranque a recepção de mensagens não está disponível, como tal foi implementado na aplicação de monitorização um buffer que permite guardar notificações cujo envio não teve sucesso. O tamanho máximo deste buffer pode ser configurado de forma a que se tenha em consideração as restrições no uso de memória das máquinas monitorizadas, no entanto, deve ter tamanho suficiente para o número de notificações que se tente enviar enquanto o intermediário não estiver disponível.

Na Tabela 4.3 e Figura 4.5 são apresentados os resultados obtidos dos testes executados num cenário onde o intermediário de mensagens foi "forçado" a estar indisponível no início dos testes. Durante o tempo em que esteve indisponível, cerca de 1 minuto e 30 segundos, as mensagens foram guardadas no buffer de memória configurado para ter o limite máximo de 100 posições. Pelos resultados obtidos constata-se o aumento de memória e do tempo de serialização e envio face aos resultados do funcionamento normal. Estes aumentos devem-se ao facto de se ter armazenado no buffer as notificações cujo envio falhou e posteriormente pelo envio de todas as notificações existentes no buffer. Nos resultados obtidos verifica-se também que o tempo de recepção do intermediário diminuiu, isto aconteceu porque quando o intermediário ficou disponível recebeu imediatamente várias mensagens seguidas. O tempo de recepção do MDB mantém aproximadamente igual ao funcionamento normal dado que o mecanismo que o intermediário usa no encaminhamento de notificações para o MDB continua a ter os mesmos atrasos iniciais de estabelecimento da conexão e sessão.

Tabela 4.3: Resultados dos testes de desempenho num cenário de indisponibilidade do intermediário

Número de Notificações Enviadas	Memória utilizada (KBytes)	Tempo de serialização e envio (mi:ss.ms)	Tempo de Recepção do Intermediário (mi:ss.ms)	Tempo de Recepção do MDB (mi:ss.ms)
100	3268	01:53.364	01:40.485	01:38.273
200	3260	03:34.629	03:23.326	03:31.578
300	3269	05:22.617	05:11.982	05:10.406
400	3268	07:06.318	06:55.335	06:53.672
500	3284	09:02.104	08:51.365	08:51.215
600	3268	10:38.092	10:28.428	10:28.526
700	3268	12:38.644	12:28.361	12:29.368
800	3260	15:45.628	15:32.734	15:38.645
900	3268	16:12.053	16:02.025	16:02.491

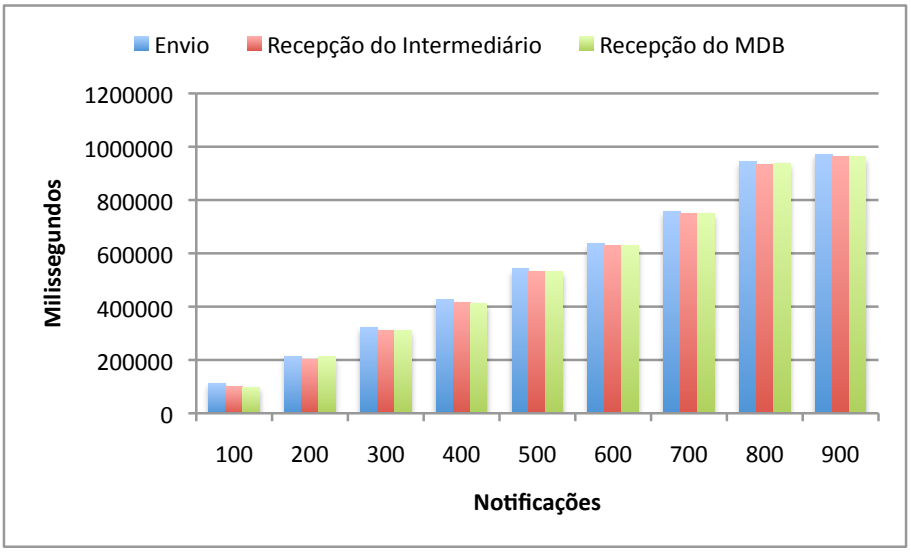

Figura 4.5: Resultados dos testes de desempenho num cenário de indisponibilidade do intermediário

Por fim, o último cenário onde existe indisponibilidade do MDB configurado no NGIN Manager é uma situação completamente transparente para o agente de monitorização, pois este continua a serializar e a enviar notificações para o intermediário. Desta forma o tamanho da memória utilizada pelo agente, o tempo de serialização e envio e o tempo de recepção do intermediário representam a mesma situação da Tabela 4.2 e do gráfico 4.4. Nesta situação o intermediário persiste todas as mensagens e base de dados, pode inclusive terminar inesperadamente, ser parado ou reiniciado que as mensagens persistidas serão enviadas quando o MDB ficar disponível. Como tal, os tempos de recepção do MDB após o restabelecimento do seu estado, são também aproximadamente iguais aos retratados na Tabela 4.2 pois, tal como no cenário de funcionamento normal onde as operações de persistência, estabelecimento da conexão e sessão inicial para o encaminhamento entre queues levam a que o MDB receba a primeira notificação com um pequeno atraso.

4.3 Sumário

Neste capítulo foram apresentados diferentes cenários de utilização da plataforma de envio de notificações, tendo em conta as necessidades de fiabilidade na monitorização dos componentes utilizados pelas operadoras de telecomunicações. Desta forma, foram apresentados três cenários de utilização e para cada um foram realizados testes de desempenho de forma a avaliar o com-

portamento da solução. Perante os resultados obtidos constatou-se que nunca houve perdas de informação e os resultados obtidos demonstram que apesar das garantias de fiabilidade a solução consegue obter bons tempos de resposta às necessidades de monitorização.

Capítulo 5

Conclusões

Ao longo deste documento foram apresentadas e explicadas todas as etapas envolvidas no desenvolvimento desta dissertação. Neste capítulo, são apresentadas as conclusões do trabalho desenvolvido, sendo feito um resumo de cada etapa e destacando as principais contribuições obtidas com a investigação, desenvolvimento e testes realizados. Por fim, serão discutidas ideias trabalhos que poderão servir para evoluir a solução actual com novas funcionalidades.

5.1 Principais Contribuições

A principal contribuição desta dissertação resultou do desenvolvimento de uma arquitectura baseada em sistemas de mensagens para intermediação de eventos de monitorização, enquadrando-se nas necessidades exigentes que é a monitorização de operações e serviços nas operadoras de telecomunicações. A contribuição desta arquitectura para a solução NGIN permite criar sistemas de comunicação de alta disponibilidade e com garantias de fiabilidade, evitando a perda de informação de monitorização crítica. Por outro lado, a nova arquitectura contribui com padrões de comunicação modulares e altamente integráveis, quer ao nível arquitectural, suportando diferentes linguagens, quer ao nível do protocolo de comunicação, integrando diferentes serviços de mensagens. As próximas secções descrevem as contribuições de cada etapa envolvida na elaboração desta dissertação.

5.1.1 NGIN Manager

Ao longo desta dissertação, o NGIN Manager foi apresentado como uma solução utilizada pelas operadoras de telecomunicações para monitorização dos serviços de rede inteligente e operacionalidade dos subsistemas da solução NGIN. Perante o crescente aumento dos serviços disponibilizados pelas operadoras de telecomunicações e número de clientes que subscrevem esses serviços, no capítulo 2 é feito o estudo e apreciação crítica das arquitectura orientada a eventos que permitam disponibilizar mecanismos de monitorização com facilidades de gestão, controlo de recursos aplicacionais e facilidades de implementação de novos módulos de monitorização. Em função das diferentes estratégias apresentadas para a concepção da nova arquitectura, as plataformas de "middleware" orientado a mensagens (MOM) destacam-se pelo tipo de arquitectura usada e versatilidade, permitindo por um lado, a comunicação com diferentes entidades de monitorização sem causar grandes interferências no processo de execução habitual e por outro lado permite uma rápida adaptação a novos contextos de monitorização através da mediação disponibilizada entre as diferentes aplicações. Esta abordagem contribuiu com a definição de uma nova plataforma de intermediação entre as aplicações monitorizadas e o NGIN Manager, garantindo requisitos de fiabilidade, desempenho e suporte transaccional na comunicação e adequando-se à heterogeneidade de aplicações e arquitecturas suportadas pela solução NGIN. As dificuldades existentes na integração tecnológica, arquitectural e protocolar não podem ser contornadas com uma imposição rígida das opções disponíveis, levando à perda de oportunidades de negócio e qualidade de software, pelo que as contribuições da nova arquitectura passam também pela definição de protocolos de comunicação e de estruturação da informação de monitorização modulares e flexíveis adequando-se facilmente à concepção de novas métricas e protocolos.

Neste contexto, a nova arquitectura apresenta-se como uma plataforma modular, que oferece funcionalidades de intermediação, integração e encaminhamento permitindo o desenvolvimento de novas tecnologias através de adaptadores e facilitando o processo de gestão da lógica de intermediação, definindo diferentes regras de encaminhamento de forma a poder diferenciar tecnologias ou componentes específicos.

5.1.2 Concepção da solução

A concepção da solução é descrita no capítulo 3 e apresenta todas as etapas envolvidas na especificação da arquitectura orientada a eventos, isto é, a integração com o NGIN Manager, definição de requisitos, desenho da arquitectura e implementação dos componentes da arquitectura. A ar-

quitectura da solução divide-se em três componentes principais, isto é, os agentes instalados nos componentes monitorizados responsáveis por serializar a informação de monitorização e proceder ao seu envio, o componente de intermediação na comunicação dos agentes com o NGIN Manager e os componentes de recepção, descodificação e processamento no servidor aplicacional do NGIN Manager.

O componente de envio de eventos contribui com a definição de uma estrutura de informação flexível que pode ser usada por diferentes protocolos de serialização, permitindo consequentemente, de acordo com as linguagens suportadas pela ferramenta usada, gerar código com estruturas de dados. De forma a possibilitar o envio de notificações foi implementada uma biblioteca em C/C++ para serialização em ASN.1 e envio de notificações para um intermediário de mensagens. O componente de intermediação de notificações é o ponto central da arquitectura e contribuiu com mecanismos que introduzem fiabilidade, disponibilidade, integração aplicacional e protocolar, encaminhamento, segurança, controlo de fluxo e permitindo implementar novos módulos no processo de intermediação. Desta forma foram implementadas duas bibliotecas, permitindo que o intermediário possa receber notificações através de um socket TCP/IP e outra que permite o controlo do intermediário pela ferramenta LSC (Local Subsytem Controller), disponibilizada pela PT Inovação. A recepção de notificações por socket permite o envio de notificações por agentes que sejam executados em ambientes com mais restrições na utilização de recursos e na utilização de APIs demasiado recentes. O controlo do intermediário permite administrar remotamente um intermediário, isto é, permite proceder ao seu arranque, paragem e definir o arranque automático em situações de paragens inesperadas do intermediário. O componente de recepção de notificações é implementado no servidor aplicacional do NGIN Manager contribuindo com uma biblioteca Java cujas classes de manipulação da informação de monitorização foi gerado a partir da especificação da estrutura de mensagens e que é invocada na recepção de uma notificação pelo Message Driven Bean instanciando um objecto que é utilizado pela lógica de processamento da informação para armazenar a informação numa data warehouse.

Desta forma, com a concepção da arquitectura orientada a eventos demonstrou-se a capacidade de adequação da plataforma a diferentes contextos, permitindo a rápida implementação de ferramentas numa arquitectura com capacidades de integração ao nível protocolar e arquitectural e com garantias de fiabilidade e alta disponibilidade. As funcionalidades de integração resultam da capacidade de suporte de diferentes serviços de mensagens como JMS, STOMP, OpenWire e através do adaptador implementado para recepção notificações por socket. A integração com arquitecturas distintas é conseguida através de APIs disponibilizadas pelo intermediário em diferentes linguagens de programação e através da possibilidade de utilização de APIs nativas para

envio de notificações por socket. As garantias de fiabilidade resultam dos mecanismos de persistência disponíveis no intermediário de mensagens, que para cada notificação recebida, procede ao armazenamento numa base de dados, eliminando-a apenas após o seu correcto encaminhamento para o Message Driven Bean do servidor aplicacional instalado no NGIN Manager. No caso de ocorrerem falhas no encaminhamento é feito um "rollback" da transacção de envio, procedendo-se periodicamente a novas tentativas de encaminhamento. A disponibilidade da arquitectura é garantida através da configuração de vários intermediários agrupados num "cluster", sendo que neste caso a indisponibilidade de um nodo é imediatamente resolvida por outro. Para além da configuração de "clusters" de intermediários foi implementada uma biblioteca que permite a gestão automática de um intermediário por parte da ferramenta LSC (Local Subsytem Controller) da PT Inovação, detectando a sua indisponibilidade e procedendo automaticamente ao seu arranque. Nesta situação, para garantir a fiabilidade durante o tempo de arranque do intermediário, foi implementado um buffer de tamanho configurável na biblioteca de serialização e envio, que armazena todas as notificações cujo envio para o intermediário falhe.

5.1.3 Cenários e Testes realizados

Os cenários de utilização e testes foram apresentados no capítulo 4, demonstrando a fiabilidade de entrega de notificações, desempenho e funcionalidade da nova arquitectura para os diferentes cenários de utilização suportados, validando os requisitos de fiabilidade e disponibilidade esperados. A situação apresentada em concreto apresentada diz respeito à monitorização de um SCF nas redes inteligentes, calculando métricas para o número de chamadas em simultâneo processadas, terminadas, taxas com o número de chamadas por segundo, estatísticas com o número de serviços invocados e número de excepções, entre outros. O primeiro cenário apresentado refere-se ao funcionamento normal esperado pela solução, em que todos os componentes se encontram activos e funcionais, isto é, um contexto em que todas as notificações enviadas pelos agentes de monitorização são devidamente encaminhadas pelo intermediário e consequentemente processadas no NGIN Manager. Os restantes cenários referem-se ao comportamento da solução num contexto, de indisponibilidade temporária do intermediário e de indisponibilidade de recepção de notificações no NGIN Manager. Em ambos os casos, nos testes realizados, não houve perdas de notificações, dado que o componente de intermediação procede à persistência das notificações recebidas e o agente de envio possui um buffer lógico para armazenamento de notificações que não consigam ser enviadas.

Desta forma, para os diferentes cenários apresentados, os testes realizados demonstraram

a fiabilidade de entrega de notificações, desempenho, funcionalidade e viabilidade da solução, tendo sido obtidos bons resultados de desempenho e acima de tudo comprovada a fiabilidade da solução. Para além de um ambiente de execução de SCFs a arquitectura foi implementada tendo em conta as necessidades de integração, pelo que se encontra preparada para monitorizar diferentes ambientes e por outro lado permite o desenvolvimento modular de novas funcionalidades adaptando-se facilmente a diferentes contextos.

Todo este trabalho envolveu decisões tecnológicas e de funcionamento lógico da arquitectura. A tecnologia utilizada pela arquitectura baseia-se em plataformas de "middleware" orientados a mensagens (MOM) dadas as suas funcionalidades de persistência e de desempenho. Como alternativa poderiam ter sido usados Web Services, no entanto, esta alternativa possui dificuldades no processo de persistência da informação em comunicações assíncronas, utilizando em muitas situações MOMs para garantir esses requisitos. Relativamente aos protocolos de serialização, foi utilizada a norma ASN.1, no entanto a arquitectura encontra-se preparada para a utilização de novos protocolos, permitindo-se inclusive a utilização de protocolos de serialização distintos para as diferentes entidades monitorizadas.

5.2 Trabalho Futuro

A arquitectura desenvolvida apresentou-se como uma solução viável, sendo que, poderão ser implementadas novas funcionalidades de forma a complementar a arquitectura base criada:

- Desenvolvimento de interfaces de administração do intermediário: Dado que o intermediário de notificações é um novo componente no conjunto do subsistema NGIN Manager, o controlo e administração deste componente terá que ser tido em conta, através da criação de interfaces que permitam a definição das regras de encaminhamento e gestão das interfaces protocolares existentes.

- Integração com os restantes componentes monitorizados pelo NGIN Manager: A solução desenvolvida foi criada tendo em conta necessidades de integração com vários componentes da rede inteligente e subsistemas da solução NGIN, como tal, esta pode integrar-se com os componentes suportados pelo NGIN Manager. Neste sentido, poderá ser feita a integração com novos componentes ou com os componentes já existentes para os quais existam necessidades de alteração do paradigma de monitorização de forma a garantir requisitos de fiabilidade.

- Integração da solução num ambiente de produção real: Dada a complexidade dos ambientes das operadoras de telecomunicações a integração da nova arquitectura nesses ambientes, poderá requerer a inclusão de novos parâmetros de configuração.

Bibliografia

[1] PT Inovação. Next generation intelligent networks. Documentação, http://www.ptinovacao.com.br/download.do?image=/pdf/f_NG_ISDF_V2b.pdf.

[2] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[3] P. Niblett and S. Graham. Events and service-oriented architecture: The oasis web services notification specifications. *IBM SYSTEMS JOURNAL*, 44(4):869–886, 2005.

[4] Ian Foster, Karl Czajkowski, Donald F. Ferguson, Jeffrey Frey, Steve Graham, Tom Maguire, David Snelling, and Steven Tuecke. Modeling and managing state in distributed systems: The role of ogsi and wsrf. *IEEE Computer Society*, 93(3):604–612, 2005.

[5] OMG. Event service specification. OMG Tech. Doc. 1.2, October 2004.

[6] OMG. Notification service specification. OMG Tech. Doc. 1.1, October 2004.

[7] S. Vinoski. Advanced message queuing protocol. *Internet Computing, IEEE*, 10(6):87–89, 2006.

[8] Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, and Kate Stout. Java message service. Technical Report 1.1, April 2002.

[9] Codehaus. Stomp protocol specification. Documentation and open source release. http://stomp.codehaus.org/.

[10] Apache ActiveMQ. Openwire. Documentation and open source release. http://activemq.apache.org/openwire.html.

[11] Yi Huang and Dennis Gannon. A comparative study of web services-based event notification specifications. *IEEE Computer Society*, pages 7–14, 2006.

[12] Roger Eggen and Suresh Sunku. Efficiency of soap versus jms. *In International Conference on Internet Computing*, pages 99–105, 2003.

[13] Sun Microsystems Kenneth Saks. Jsr 318: Enterprise javabeans. Technical Report 3.1, December 2009.

[14] AMQP Working Group. Advanced message queuing protocol, 2010. `http://www.amqp.org`.

[15] Matt Welsh, David Culler, and Eric Brewer. Seda: an architecture for well-conditioned, scalable internet services. *SIGOPS Oper. Syst. Rev.*, 35(5):230–243, 2001.

[16] B. Michelson. Event-driven architecture overview: Event-driven soa is just part of the eda story. White paper, Patricia Seybold Group, June 2006. Available online (9 pages).

[17] Alejandro P. Buchmann and Boris Koldehofe. Complex event processing. *it - Information Technology*, 51(5):241–242, 2009.

[18] Rainer Von Ammon, Christoph Emmersberger, Florian Springer, and Christian Wolff. Event-driven business process management and its practical application taking the example of dhl, 2008.

[19] Daniel Gyllstrom, Eugene Wu, Hee jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. Sase: Complex event processing over streams. In *In Proceedings of the Third Biennial Conference on Innovative Data Systems Research*, 2007.

[20] JBoss. Jboss jms performance benchmark, 2006. `http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossJMSNewPerformanceBenchmark`.

[21] Apache Software Foundation. Apache activemq reference documentation. `http://activemq.apache.org`.

[22] Red Hat. Inc. jboss hornetq. `www.jboss.org/hornetq`.

[23] iMatix. Openamq. `www.openamq.org`.

[24] Sun Microsystems. Open message queue. `www.openamq.org`.

[25] Apache Software Foundation. Qpid. `http://qpid.apache.org`.

[26] SpringSource. Rabbitmq. `http://www.rabbitmq.com/`.

[27] Adra Al Mosawi, Liping Zhao, and Linda Macaulay. A model driven architecture for enterprise application integration. volume 8 of *Proceedings of the 39th Hawaii International Conference on System Sciences*. IEEE Computer Society, 2006.

[28] F.B. Vernadat. Interoperable enterprise systems: Principles, concepts, and methods. *Annual Reviews in Control*, 31(1):137 – 145, 2007.

[29] M. P. Papazoglou and W. J. Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, July 2007.

[30] ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[31] ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

[32] ITU-T Recommendation X.691 (2002) | ISO/IEC 8825-2:2002, Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).

[33] ITU-T Recommendation X.693 (2002) | ISO/IEC 8825-4:2002, Information technology - ASN.1 encoding rules: XML Encoding Rules (XER).

[34] Steven Legg. RFC 3641: Generic string encoding rules (gser) for asn.1 types, October 2003.

[35] Mike Eisler. RFC 4506 xdr: External data representation standard, May 2006.

[36] Google. Protocol buffers: Google's data interchange format. Documentation and open source release. `http://code.google.com/p/protobuf/`.

[37] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. Thrift: Scalable cross-language services implementation, April 2007.

[38] Avro 1.3.3 overview, 2010. `http://avro.apache.org/docs/current/index.pdf`.

[39] Eishay Smith. Comparing varius aspects of serialization libraries on the jvm platform. Google code. `http://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking`.

[40] Eishay Smith. jvm-serializers. GitHub Inc. `http://wiki.github.com/eishay/jvm-serializers/`.