INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de
Eletrónica e Telecomunicações e de Computadores

Utilização de eventos aplicacionais para análise de negócio em tempo real

VITOR MIGUEL LOURENÇO ESTEVES PAULINO
(Bacharel Pré-Bolonha)

Trabalho Final de Mestrado para a obtenção de grau de Mestre,
em Engenharia Informática e de Computadores

Orientador(es):
Doutor Nuno Miguel Soares Datia
Doutor Artur Jorge Ferreira

Júri:
Presidente: Doutor Manuel Martins Barata
Vogais:
Doutora Cátia Raquel Jesus Vaz
Doutor José Manuel de Campos Lages Garcia Simão
Doutor Nuno Miguel Soares Datia

Outubro de 2017

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

API – *Aplication programming interface*

CEP – *Complex Event Processing*

CPE – *Customer Permisses Equipment*

EPG – *Electronic programming Guide*

ETL – *Extract Transform Load*

EDA – *Event Driven Arquitecture*

FTP – *File Transfer Protocol*

HTTP – *Hypertext transfer protocol*

IIS – *Internet information services*

JSON – *Javascript object notation*

JSON-LD *Javascript object notation for linking Data*

KIF – *Knowledge interchange format*

KPI – indicadores de desempenho operacional

LOB – *Line of business*

OLAP – *Online Analytical processing*

OTT – *over the top*

OMG - *Object Management Group*

OO – *Object oriented paradigm*

OIL – *Ontology Inference Layer*

OWL - *Web* Ontology Language

RDF - *Resource Description Framework*

SCEP – *Semantic Complex Event Processing*

SHOE - *Simple HTML* Ontology *Extensions,*

STB – *Set-top boxes*

SPARQL - *SPARQL Protocol and RDF Query Language*

SWRL - *A Semantic Web Rule Language Combining OWL and RuleML*

SSE - *Semantic Structured Events*

UML – *unified modeling language*

VOD - *Video on demand*

XOL - *XML-Based* Ontology *Exchange* Language,

XML – *Extended Markup language*

ZIP – Formato de compactação de arquivos

Resumo

Os dados recolhidos nos sistemas da NOS Inovação, para fins de processamento analítico, apresentam limitações ao nível da estrutura e vocabulário da mensagem. Este trabalho tem como objectivo caracterizar o formato e conteúdo das mensagens de logs que existem actualmente, e melhorar a sua estrutura e vocabulário permitindo simplificar e homogenizar o processo analitico. A identificação e definição do conteúdo das mensagens são a base para a definição de uma representação de conhecimento de eventos aplicacionais. Esta representação de conhecimento mento descritiva consiste na caracterização da TBOX e de ABOX no dominio da organização. A TBOX consiste na definição da ontologia de domínio deste trabalho, que especifica formalmente a estrutura e vocabulário das novas mensagens de log. Esta definição formal irá permitir que as mensagens sejam usadas para processamento de eventos complexos (CEP), na realização de operações de composição, agregação e interrogação. Para validação da estrutura e vocabulário da ontologia, foi desenvolvida uma biblioteca aplicacional. Esta biblioteca é composta por um modelo de dominio que representa a ontologia e ainda por mecanismos que permitem a sua instanciação, formatação e envio para processamento analitico em tempo real. As mensagens geradas por esta biblioteca acabam por representar a ABOX na descrição de eventos aplicacionais enquanto dominio de conhecimento deste trabalho. O facto de se tratar de uma biblioteca independente dos serviços da NOS Inovação, permite que qualquer sistema use esta biblioteca. Uma vez que a representação de conhecimento de eventos aplicacionais é feita pela ontologia (TBOX) e pela biblioteca (ABOX), garantimos que outras bibliotecas possam ser criadas baseadas na ontologia, gerando mensagens de logs com um vocabulário transversal à organização, e com uma estrutura de relações homogenea. A utilização da biblioteca permite que todos os serviços da NOS Inovação compativeis possam gerar mensagens de log que são interpretadas por humanos e também por processos automáticos de igual forma e sirva de base para a criação de bibliotecas para outros ambientes de execução, como por exemplo dispositivos móveis, ou CPEs.

Palavras-chave: Eventos, Ontologias, Processamento de eventos complexos.

Abstract

At NOS Inovação there is a need to improve the data collection process that characterizes the occurrence of customer use cases that occur in the organization's systems. The approach to this problem is to identify the concepts considered relevant to the organization by studying the current system of data collection used for analytical process and also analyzing the main use cases. The identification and definition of these concepts are the basis for the definition of a domain ontology that allows the definition of processes for processing complex events (CEP) with which it is possible to perform composite, aggregation and interrogation operations based on a normalized structure and with semantic meaning. CEP contains mechanisms to detect complex patterns and automatically generate responses when patterns are detected on recorded events. The demonstration of the use of this ontology consisted in the development of an application library whose domain is composed by the concepts defined in the ontology. The developed application library allows the registration of the application events in a homogeneous structure defined by the ontology and send this record in a standard format to a real-time event processing server.

keywords: Events, Ontology, Complex events processing.

1 Introdução

No contexto de uma organização, os sistemas informáticos têm como objectivo suportar o funcionamento de processos e serviços. Esses sistemas ganham uma maior relevância quando também participam no processo de evolução do negócio, ao contribuir na construção de *key performance indicators* (KPI) [1]. Uma das fontes de dados de onde as empresas obtêm informação sobre o consumo dos seus produtos para a medição de KPIs são os próprios sistemas que as organizações desenvolvem ou adquirem, designados por aplicações *Line of Bussiness* (LOB). O agregado desses sistemas e aplicações formam dentro de uma empresa o que se designa por ecossistema aplicacional de uma organização. Numa organização de grande escala, a extração de dados para o enriquecimento do conhecimento sobre o seu negócio é um processo que gera grandes quantidades de dados que não ficam imediatamente disponíveis para serem utilizados na extração de conhecimento [2]. Existem vários desafios que têm de ser abordados, relacionados com o volume de dados recolhidos, a variedade de sistemas fonte e de formatos de representação dos dados, bem como a velocidade de processamento desses dados para garantir a produção de conhecimento em tempo útil. A gestão de dados com estes desafios caracteriza-se pela regra dos três V: volume de dados, velocidade de processamento de dados e variedade dos dados como sendo um problema de *big data* [3, 4].

Extrair de um sistema a informação de que ocorreu um caso de utilização de negócio e caracterizar essa ocorrência não é um processo fácil, uma vez que os sistemas LOB não se focam nesse objectivo, mas sim em prestar serviço aos clientes. Os modelos aplicacionais usados na implementação desses serviços consistem em persistir e manter o estado do seu domínio aplicacional actualizado, Por outro lado, devem descrever o processo que levou ao atual valor tomado pelo estado do domínio aplicacional. Para essa tarefa existem mecanismos que podem ser adicionados aos sistemas, transversais a requisitos de negócio, que permitem registar a ocorrência de rotinas aplicacionais e assim auxiliar o processo de caracterização da ocorrência de casos de uso aplicacionais. Esses mecanismos designam-se por *Logs* ou *Tracing*. Estes dois mecanismos são usados para monitorização do bom funcionamento das aplicações, através de registo de ocorrência de rotinas de código pelo seu nome com sucesso ou erro, quer seja para monitorização de desempenho na utilização de recursos físicos, tais como memória ou CPU. No entanto, são limitados no que se refere à necessidade de conseguir caracterizar a ocorrência de um caso de uso aplicacional. Um dos RFCs conhecidos e usados é o syslog[1]. Syslog é um standard de logging de mensagens de texto que segue uma convenção com base em palavras-chaves para descrever partes do log, mas não formaliza uma estrutura explícita. A falta de formalidade na definição conceptual

[1] https://tools.ietf.org/html/rfc5424

do que se pretende registar numa escrita de log pode tornar complexo o seu reaproveitamento para processamento analítico resultando muitas vezes em implementações à medida em processos de extração de dados desses logs dentro das organizações.

No software que é desenvolvido na NOS esse problema também existe. Associados a este processo existem pessoas envolvidas com três tipos de papéis: (i) Definição do produto (ii) Desenvolvimento de software para dar suporte a esse produto (iii) e por último análise dos dados que esse software gera, por forma a avaliar o sucesso ou insucesso dos produtos, apresentado na Figura 1. A comunicação que existe entre a definição do produto e as equipas de desenvolvimento consiste em métodos bem conhecidos na organização num formato que é do conhecimento de ambas as partes onde ideias e as especificações são transmitidas de forma normalizada. Tal processo também acontece entre as pessoas do produto e os analistas de dados. Mas para que os analistas dos dados consigam fazer o seu trabalho, os dados que as aplicações geram para eles interpretarem têm de estar também num formato bem conhecido entre as equipas de desenvolvimento de software, e os analistas de dados, e isso não acontece actualmente.

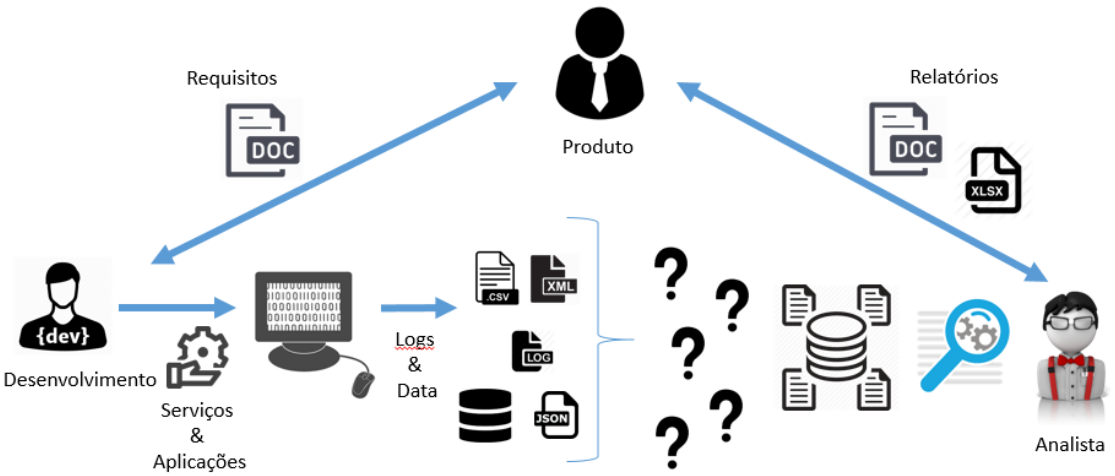

Figura 1 - Problema actual

Numa organização onde o número de sistemas está sempre a crescer, assim como o número de clientes, os três V do *big data* são valores que estão sempre a aumentar. Por isso é importante que as aplicações LOB que participam no processamento analítico o façam num formato formal explícito que represente os conceitos que se considerem relevantes transversal a todos os sistemas que geram dados para esse processo e que seja conhecido por todos os envolvidos na construção de um produto. A ocorrência de um caso de uso aplicacional, pode designar-se como a ocorrência de um evento aplicacional. Segundo a definição de David Luckham, um evento é um objecto que representa uma atividade que aconteceu no passado ou que está prestes a acontecer [8]. Um evento pode ser agregado com outros eventos construindo dessa forma eventos complexos relacionados entre si por motivos causais ou temporais.

1.1 Objetivos

Este trabalho tem como objetivo:

1. Especificação de uma ontologia de domínio de eventos aplicacionais
2. Desenvolvimento de uma biblioteca que permita o registo de eventos aplicacionais com a estrutura definida pela ontologia
3. Desenvolvimento de uma biblioteca que permita o envio do evento aplicacional registado para um servidor de processamento analítico

1.2 Solução proposta

Neste trabalho definiram-se os conceitos e estrutura de uma representação do conhecimento de eventos aplicacionais para homogenesizar o processo de extração de dados dos serviços e a sua posterior utilização para o processamento analítico de dados dos sistemas LOB da NOS. A conceptualização dos eventos aplicacionais foi definida para ser transversal ao domínio de negócio da NOS Inovação, capaz de registar acontecimentos sobre conteúdos *electronic programming guide* (EPG) e *vídeo on demand* (VOD). Para atingir esse objectivo, este trabalho define uma ontologia que formaliza eventos aplicacionais composto por diferentes domínios de conhecimento. Assim, será possível a criação de mecanismos e ferramentas que utilizem essa informação e interpretem os dados da mesma forma, transversalmente entre sistemas, quer sejam LOB quer sejam para o processamento analítico, permitindo resolver o problema de interpretação identificado na Figura 1 através da criação de um meio de comunicação homogénea entre os dois papéis envolvidos na definição de *software* sugerido pela Figura 2

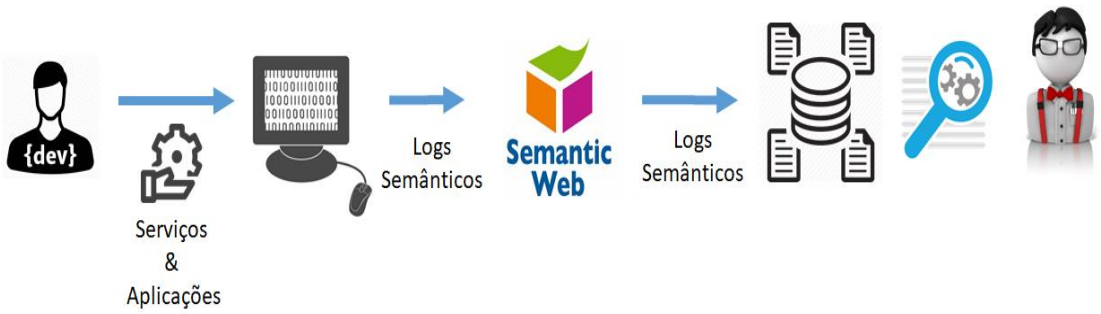

Figura 2 - Abordagem da solução

A solução para cumprir os objetivos enumerados, foi desenvolvida em três fases. A primeira fase consistiu na caracterização do processo analítico actual, na identificação das limitações existentes nesse processo, sendo feito um levantamento das aplicações LOB da NOS Inovação que suportam o serviço TV nas *set-top boxes* (STB) e aplicações *over the top (*OTT) alvo de estudo para o registo de eventos aplicacionais. Pretende-se identificar os sistemas que são relevantes no registo de eventos aplicacionais para processamento analítico. Após a identificação das limitações actuais, e dos sistemas existentes, iniciou-se a segunda fase do trabalho. Esta consistiu na criação de uma especificação formal da estrutura dos eventos aplicacionais, utilizando como referência,

ontologias já aplicadas ao domínio dos eventos. Na terceira fase do trabalho foi aplicada com sucesso a ontologia definida na fase 2. Esta foi implementada e utilizada numa solução de registo e envio de eventos aplicacionais em tempo real através de um conjunto de protocolos. A demonstração da utilização da ontologia consiste num conjunto de bibliotecas modulares, reutilizáveis, compatíveis com os sistemas aplicacionais identificados na fase 1. Estas bibliotecas terão a responsabilidade de: (i) Detectar e registar a ocorrência de eventos aplicacionais a partir das rotinas de código existentes, (ii) Normalizar os dados recolhidos para um conjunto de valores descritos na ontologia, (iii) serializar para um formato standard a descrição dos eventos, (iv) publicar esses eventos através de um protocolo de comunicação.

1.3 Organização do documento

Este documento está organizado da seguinte forma. No capítulo 2 é descrito o problema que este trabalho aborda e para o qual pretende apresentar soluções. No capítulo 3 aborda-se o estado da arte, sobre a teoria da representação de conhecimento através de ontologias. No capítulo 4 apresenta-se uma descrição detalhada da construção da ontologia, e no capítulo 5 a descrição das bibliotecas e seu funcionamento para a recolha de eventos aplicacionais. No Capitulo 6 serão apresentadas as conclusões deste trabalho bem como trabalho futuro a desenvolver.

2 Formulação do problema

A formulação do problema neste trabalho, foca-se na análise das principais características do processo de recolha dos dados dos serviços e sistemas LOB, quer seja na sua forma, quer seja no seu conteúdo. A secção 2.1 apresenta uma visão geral do processo actual da recolha de logs até ao processamento analítico desses mesmos logs. A secção 2.2 é focada em descrever a estrutura que atualmente é aplicada no registo dos Logs. Na secção 2.3 a análise é focada mais no conteúdo do que é escrito no log, nomeadamente o vocabulário utilizado para descrever os dados recolhidos. Na secção 2.4 aborda-se o problema da falta de semântica do log actual e da importância da existência dessa mesma semântica. Na secção 2.5 pretende-se evidenciar a volumetria de dados que é recolhida diariamente e a importância que a representação homogénea do log tem face a essa característica. Na secção 2.6 descreve-se o processo actual de envio dos logs para processamento analítico e quais as limitações que isso coloca para a utilização de todos os dados recolhidos em tempo útil e com sucesso. Por último na secção 2.7 é feita uma apresentação da problemática que existe no processo da normalização quando se pretende ter informação ou conhecimento temporalmente mais perto da ocorrência dos eventos.

2.1 Processo analítico actual

A análise de *logs* providencia informação relevante sobre os processos que estão em execução no ecossistema aplicacional da NOS Inovação. Esta análise ajuda a identificar o bom funcionamento dos sistemas através de monitorização dos erros que acontecem, mas também na análise do consumo de diferentes produtos que a NOS Inovação oferece aos seus clientes. Esta análise pela sua importância que tem para gerar valor para a organização começa a ser limitada, dadas as suas características actuais. O processamento de ficheiros de texto semiestruturados gerados de diferentes fontes de processamento temporalmente desfasados da sua ocorrência é um processo que não escala. Por cada sistema novo que surja na organização um novo formato de *log* é criado e enviado para o processamento analítico, onde este por sua vez terá de ser alterado para pode processar esse *log*. O processo de recolha de informação para processamento analítico da NOS Inovação consiste no processo apresentado na Figura 3.

Este processo começa no ecossistema aplicacional onde são realizadas escritas de duas fontes identificadas no ponto 1 da Figura 3:

1. Ficheiros de texto pelo servidor Web, designados por *logs* de IIS, onde os serviços são instalados
2. Ficheiros de *logs* aplicacionais.

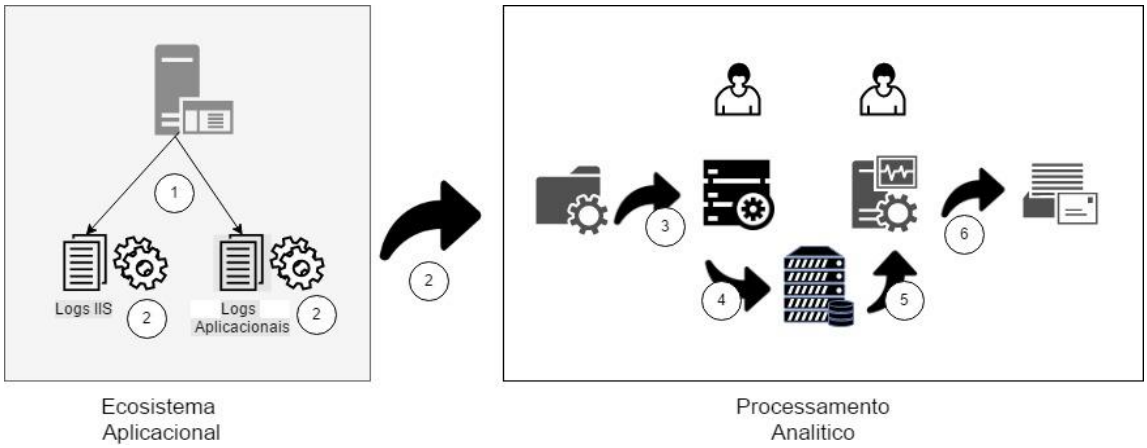

Figura 3 - Processo analítico actual

Os ficheiros de IIS estão configurados como apresentado na Figura 4. A configuração deste log é orientada à produção de logs de todas as aplicações para o mesmo ficheiro agregando dessa forma num intervalo de tempo configurado, actualmente de 1h, as acções que acontecem em todos os sistemas. O formato dos logs utilizado, W3C[2], produz informação associada aos pedidos HTTP e os campos usados actualmente estão na Tabela 1.

Tabela 1 - Campos do log IIS utilizado na NOS Inovação

Nome do campo	Descrição
date	Data de ocorrência
time	Hora da ocorrência
s-ip	O IP do servidor onde o log foi gerado
cs-method	O método HTTP
cs-uri-stem	Endereço relativo da aplicação acedido
cs-uri-query	Query que o cliente está a tentar realizar
s-port	O porto que esta configurado para o serviço
cs-username	Utilizador autenticado que acedeu ao servidor
c-ip	IP do cliente que fez o pedido
cs(user-agent)	A aplicação cliente que foi usada para fazer o pedido
cs-host	Nome do hospedeiro do serviço presente no cabeçalho HTTP.
sc-status	*Status* HTTP
sc-substatus	Sub *status* do código de erro
sc-win32-status	*Status* com um código do Windows
sc-bytes	Numero de bytes enviados pelo serviço na resposta
time-taken	A duração da execução da acção

[2] https://msdn.microsoft.com/en-us/library/windows/desktop/aa814385(v=vs.85).aspx

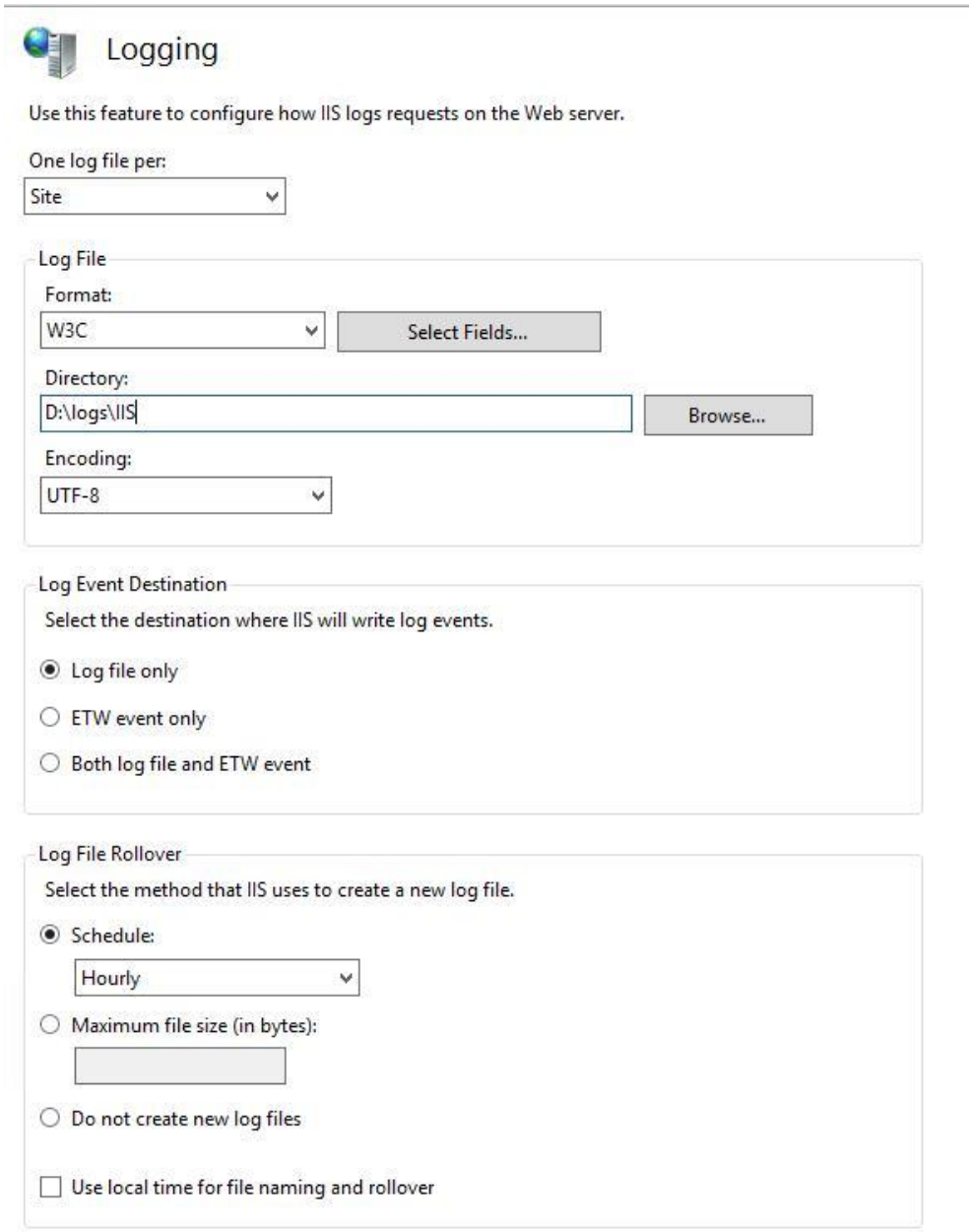

Figura 4 – Configuração de logs de IIS

Os ficheiros dos *logs* aplicacionais, registam a ocorrência da execução de uma rotina de código. Ao contrário dos logs do IIS, este log consiste em recolher informação aplicacional que caracteriza a ocorrência do caso de utilização da aplicação. O *log* aplicacional é escrito com uma estrutura própria criada na organização com a qual se pretende registar dados de negócio, tais como, quem foi o cliente que fez o pedido, qual foi a acção que foi realizada, qual foi o resultado, e tratando-se de um negócio que se baseia na utilização de equipamentos por parte dos clientes o log aplicacional também regista essa informação, na secção 2.2 será feita uma analise mais detalhada a este log.

Os ficheiros onde são persistidas as entradas de *logs* estão localizados nos servidores onde os sistemas estão em execução. Para que os ficheiros sejam utilizados no processo analítico, estes

têm de ser enviados para os servidores que persistem todos os ficheiros das aplicações. O envio dos ficheiros para esses servidores é realizado por tarefas calendarizadas no sistema operativo que recolhem os ficheiros para um repositório de ficheiros de texto (2) existente no data center da NOS.

O próximo passo consiste num processo ETL responsável por processar os ficheiros (3). Este processo, lê de cada um dos ficheiros os dados relevantes, e coloca-os numa base de dados. Estes passos de normalização são configurados por uma pessoa que conhece a estrutura dos ficheiros de texto que foram enviados e que sabem quais os campos relevantes que devem ser recolhidos dos ficheiros.

O resultado desse processo de normalização resulta no registo em tabelas factuais e de dimensões numa base de dados OLAP (4). Por cada vez que existe necessidade de apresentar dados de negócio sobre os dados guardados, um utilizador, acede ao serviço de geração de relatórios, e inicia o processo de geração de relatórios de onde são produzidas folhas Excel (6) que são posteriormente enviadas por E-mail.

Na evolução deste modelo de extração de informação dos sistemas aplicacionais para um sistema *big data*, este processo apresenta limitações para a necessidade que existe em recolher uma maior volumetria de dados de serviços aplicacionais cada um com o seu domínio de informação próprio a velocidades cada vez maiores. Outra limitação consiste na capacidade de produzir resultados de análise sobre os dados recolhidos num intervalo de tempo suficientemente curto para conseguir realizar acções pró-activas em vez de acções reativas. Numa análise feita sobre este processo identificaram-se as limitações que vão ser alvo de trabalho neste estudo:

1. *Logs* semi-estruturados
2. Vocabulário não formalizado
3. Semântica nos *logs*
4. Volumetria dos *logs*.
5. Envio dos logs para processamento analítico
6. Processo de normalização

2.2 *Logs* semi-estruturados

No desenvolvimento de serviços aplicacionais existe a necessidade de garantir que a aplicação escreva para um repositório de ficheiros a informação de *Logging* e de *Tracing*, por forma a observar o desempenho dos servidores que suportam esses serviços e o comportamento da lógica implementada validando se não existem problemas. Logging consiste no registo da ocorrência de situações na aplicação com informação a mais detalhada possível com diferentes níveis de severidade, tais como Informação, Aviso, Erro e Fatal. O registo destas situações nos seus diferentes níveis de severidade é definido pela equipa de desenvolvimento do sistema, onde a

preocupação é disponibilizar ao leitor desses Logs informação que lhe permita identificar: (i) Severidade da ocorrência do Log (ii) Qual a data e hora a que aconteceu (iii) em que componente do sistema é que aconteceu (iv) e o que é que aconteceu em formato texto livre por vezes multi linha.

Tracing consiste em registar o fluxo de execução de uma ou várias *threads* pelos componentes, onde o foco do registo desta informação consiste em identificar:

1. Nome do componente.
2. Nome da rotina.
3. Parâmetros e seus valores.
4. Duração da execução da rotina.

A estrutura dos *logs* actuais produzidos para a finalidade de processamento analítico baseiam-se numa estrutura desenvolvida especificamente para este fim baseada em regras de processamento de texto dependendo de caracteres ASCII que separam as várias secções da linha do Log. Estes logs ocupam uma linha de texto, com toda a informação relevante ao processamento analítico e são constituídos por três partes:

1. DataHora, Nome do *Logger*, Severidade, *Thread* Id
2. Pares chave valor do log aplicacional onde se encontram os parâmetros das acções
3. Campos a serem usados pelo processamento analítico separados por ponto e vírgula.

O exemplo apresentado na Figura 5 apresenta uma entrada de um log onde se registou que o cliente acedeu à lista de aplicações disponíveis para a *set-top box*.

O log possui informação, mas não está estruturada. Apesar do log registar quando, onde e em que aplicação é que aconteceu, a estrutura não garante o significado do valores. Para campos relevantes ao processamento analítico não existe uma descrição tipificada dos campos registados, existe sim uma dependência directa d a ordem com que os campos são escritos para o seu significado, facilitando a que ocorram erros na interpretação dos valores e do seu significado, sendo por isso também uma estrutura que não suporta facilmente a mudança dos parâmetros.

Em acções onde existem parâmetros de entrada como o exemplo apresentado na Figura 6, da compra de um conteúdo no videoclube, este log apresenta outro problema relacionado com os dados recolhidos. Para identificar o sujeito de uma acção, entidade do domínio aplicacional, embora essa informação exista, ela não exposta num campo dedicado. No processo actual o que aconteceu é representado pelo URI completo perdendo-se indicação que que a acção Alugar aconteceu sobre o conteúdo do vídeoclube com o Id vod@000259_LUS_JUMPER_61554000_. Para conseguir extrair essa informação do *log*, o processo de normalização do processo da Figura 3 terá de ser enriquecido com mais um passo de processamento para este URI em concreto.

```
[2017-04-24 13:53:32,963|NOS.Logging.log4net.PostSharp.FEFormat.EntryPointLoggingAttribute|INFO |220]
[
 { expand=(null),
   skip=(null),
   log4net:UserName=IIS APPPOOL\NextGenAppPool,
   log4net:Identity=,
   expandVersion=(null),
   reqId=(null),
   expandChildren=(null),
   log4net:HostName=SVLNDIDFE27,
   top=(null)
}]
:
00d0375d1031;
SVLNDIDFE27;
ApplicationsController.GetAppRootItems;
GET /NextGen.FE/api/apps/root/items;
1.0.0-RC1b;
16;
[X-Core-UserType:profile|
X-Core-UserId:?|
X-Core-DeviceId:00d0375d1031|
X-Core-DeviceType:stb|
X-Core-AccountId:s801004230|
X-Core-Language:por|
X-Core-AppId:NEXTGEN_E|
X-Core-AppVersion:1.0.0-RC1b|
X-Core-AppFeatures:?|
X-Core-UserProfileId:501113|
X-Core-Profile:?|
X-Core-UserDisplayName:?|
X-Core-UserSessionId:?|
X-Core-NetworkId:42833|
X-Core-VideoSessionId:?|
X-Core-ClientIp:10.147.26.34||
X-Core-Username:?];
false||
```

Figura 5 - Log Semi-estruturados de listagem de aplicações

```
[2017-04-24 14:01:00,454|NOS.Logging.log4net.PostSharp.FEFormat.EntryPointLoggingAttribute|INFO |220]
[{
log4net:UserName=IIS APPPOOL\NextGenAppPool,
productInfoType=(null),  actionToEx=(null), log4net:Identity=, productTitle=(null), macId=(null),
assetId=(null), serviceAccountId=(null), userId=(null), voucherCode=(null),actionParameters=(null),
productGenre=(null), reqId=(null), log4net:HostName=SVLNDIDFE27, viewLenght=(null), contentId=(null),
 user=(null), purchasedVod=(null), price=(null)}]
:00d0375d1031;
SVLNDIDFE27;
ContentsController.SubmitAction;
POST /NextGen.FE/api/contents/vod@000259_LUS_JUMPER_61554000_/actions/purchase_tvod/submit;
1.0.0-RC1b;
489;
[X-Core-UserType:profile|
X-Core-UserId:?| X-Core-DeviceId:00d0375d1031| X-Core-DeviceType:stb| X-Core-AccountId:s801004230|
X-Core-Language:por| X-Core-AppId:NEXTGEN_E| X-Core-AppVersion:1.0.0-RC1b| X-Core-AppFeatures:?|
X-Core-UserProfileId:501113| X-Core-Profile:?| X-Core-UserDisplayName:?| X-Core-UserSessionId:?|
X-Core-NetworkId:42833| X-Core-VideoSessionId:?| X-Core-ClientIp:10.147.26.34| X-Core-Username:?];
false||
```

Figura 6 - Log semi-estruturado de compra no videoclube

2.3 Vocabulário não formalizado

Nos *logs* apresentados, constata-se que conceitos do domínio aplicacional transitam para o processo analítico. Os *logs* que são extraídos das aplicações, são enviados para o domínio do processamento analítico que usam definições de domínios contextualizadas às aplicações. Essa passagem de domínios limita a escalabilidade do processo analítico face a novos serviços que

gerem informação desse mesmo domínio mas que o define com outro nome. Considere-se o caso de utilização de aluger do vídeoclube apresentado na Figura 7.

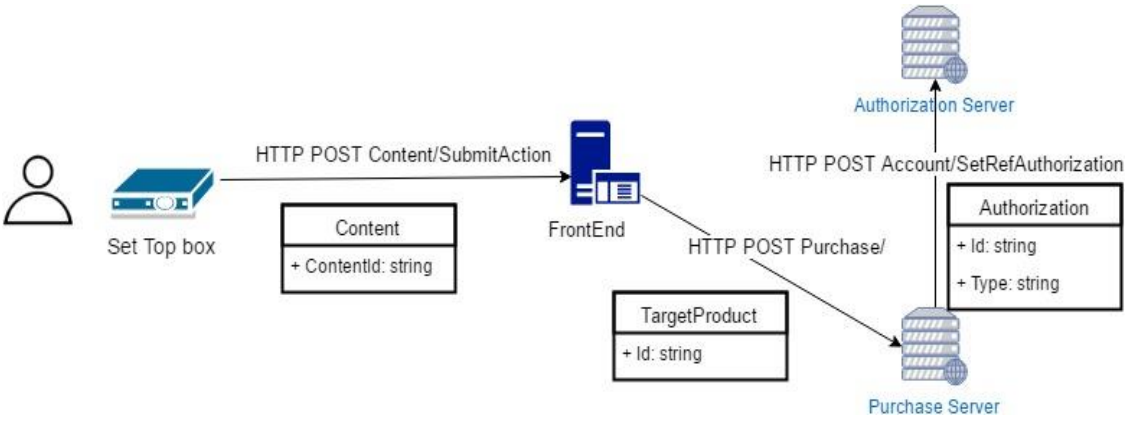

Figura 7 - Representação heterogénea dos mesmos conceitos

Neste caso de negócio existe interação entre três sistemas que recebem nos pedidos HTTP uma entidade de domínio que representa o conteúdo do vídeo clube que foi comprado, mas em cada um destes sistemas a definição do domínio difere no nome e na estrutura. O sistema FrontEnd recebe o pedido da set-top box para realizar a compra. O domínio que caracteriza este pedido consiste no tipo designado por Content identificado com o seu Id no campo *contentId*. De seguida o *FrontEnd* envia um pedido ao serviço que gere as compras dos clientes representado pelo objecto de domínio *Purchase* que tem associado a si a caracterização do conteúdo a ser comprado num objecto do Tipo *TargetProduct*. Por último depois de validada as regras de negócio sobre a compra que o cliente está a realizar, é enviado o pedido de colocar a autorização de visualização desse conteúdo no serviço que gere o portfolio do cliente. Esse pedido ao chegar ao servidor, representa esse conteúdo como sendo o objecto do tipo *Authorization*. Neste caso de utilização, cada um dos sistemas a enviar informação para o processamento analítico vai reportar diferentes nomes que caracterizam a mesma entidade de negócio.

As definições de conceitos de negócio são mais abrangentes do que o sistema que os implementa. São conceitos que estão presentes em diferentes âmbitos da organização. Existe vantagem em expor num componente comum aos serviços aplicacionais e ao processamento analítico os conceitos de negócio presentes na organização. Desta forma consegue-se abstrair o sistema do processamento analítico dos sistemas que trabalham com esses conceitos.

2.4 Semântica nos Logs

A escrita dos *logs* semi estruturada apresentada anteriormente não suporta semântica associada ao registo do *log*. Nos *logs* produzidos actualmente, não existe significado associado a cada um dos valores e não é representada a relação entre os diferentes valores. Não existe nenhum modelo conceptual que tipifique o significado dos campos e defina uma relação entre eles. Actualmente o log consiste numa linha de texto livre que segue uma regra na ordem dos campos, sem garantias

que o que fica escrito numa determinada posição da linha de texto é o valor, não exprimindo o significado desses valores.

2.5 Volumetria dos Logs

Métricas retiradas recentemente de um dos sistemas alvo de estudo deste trabalho, revelam que diariamente esse sistema produz 20G de ficheiros que são escritos para dois destinos: (i) ficheiros locais do servidor WEB (ii) e ficheiros de de logs aplicacionais. Esta volumetria de dados dispersa em diferentes repositórios apresenta um problema que é o de colocar todos esses ficheiros íntegros e disponíveis para processamento analítico em tempo útil. Com o aparecimento de novos serviços e sistemas que suportem novos casos de utilização ou novas áreas de negócio, a extração de *logs* desses serviços e sistemas para processamento analítico irão impactar o processo actual. O envio dos *logs* iria demorar mais tempo, o processo de normalização terá de processar o dobro dos ficheiros, o que resultará num processo mais lento na recolha de valor na informação recolhida.

2.6 Envio dos Logs para processamento analítico

A utilidade dos ficheiros com informação de *log* e de *Tracing* está contida no tempo, ou seja, só se for detectado um problema no instante de tempo actual é que esses logs vão ser a base para análise do problema. Com o passar do tempo esses ficheiros, e por motivos de gestão de espaço físico esses logs perdem a sua validade e são descartados e até mesmo apagados do sistema de ficheiros para onde foram criados. A utilização desses *logs* para processamento analítico faz com que esses *logs* tenham ser guardados durante mais tempo, criando problemas de gestão na volumetria dos dados já identificados anteriormente. A persistência destas escritas acontece para repositórios de ficheiros desconectados do processamento analítico, mas com limitações na capacidade de volume de dados a serem guardados, sendo necessário num intervalo regular a copia desses *logs* para o servidor de processamento analítico para serem utilizados nesse mesmo processo, sendo descartados após um período de tempo.

Os *logs* recolhidos são agrupados por dia e enviados em ficheiros ZIP por FTP para o servidor de processamento analítico. Esta é uma operação que consome tempo, e por vezes resulta em ficheiros ZIP corrompidos só detetados quando se tenta fazer descomprimir os ficheiros no destino, que obriga a recomeçar todo o processo de agregação de ficheiros de log que faz com que se consuma mais tempo. Outro especto que torna difícil a manipulação dos ficheiros prende-se com o formato com que o ficheiro foi gravado na fonte que é incompatível para leitura no destino.

2.7 Processo de normalização

O processo de normalização é um dos processos com maior valor nesta cadeia de passos até à produção de conhecimento. É nesta fase que os ficheiros semi-estruturados de *logs* e de *tracing* são analisados, processados, filtrados, transformados, onde acabam por dar origem a estruturas

de dados normalizadas que representam informação. A falta de estrutura nos logs dificulta o processo de normalização, obrigando à construção de processos especializados para a normalização dos dados. O facto de os *logs* serem extraídos de diferentes fontes de dados e outro factor que introduz variedade nos formatos a tratar. A técnica usada actualmente na NOS Inovação para atingir a estrutura normalizada consiste em realizar o parse ao texto com base em regras estáticas à estrutura que o ficheiro deverá apresentar contextualizadas no sistema origem. Caso não seja possível, a entrada de Log fica inválida para processamento sendo descartadas potenciais linhas de logs com valor para o processamento analítico.

Da análise feita neste capítulo, é relevante que a extracção de dados continue a ocorrer junto dos sistemas que suportam as áreas de negócio da NOS Inovação, sendo necessário melhorar o registo e envio do *log* para processamento analítico. Na secção 2.1 identificou-se que existem diferentes intervenientes na manipulação do *log* que é criado, desde a sua criação até á sua utilização no processamento analítico. É necessário definir um *log* com uma estrutura cujo formato seja transversal onde quer que seja usado, facilitando dessa forma o processamento analítico, uma vez que passa a processar *logs* com uma estrutura que não muda independentemente da sua origem. Esse log terá de representar os diferentes conceitos atualmente registados de forma coesa e modular, onde cada um dos campos tem o seu significado bem definido, ao contrário do exemplo do URI na secção 2.2, de onde são extraídos múltiplos valores no processamento analítico. Outro ponto a melhorar é o vocabulário presente nos *logs*. Os termos e conceitos de negócio que estão presentes na documentação gerada pelo departamento do produto, são bem conhecidos e representados nos serviços e sistemas da organização, mas não são aplicados na escrita dos *logs*. Para ultrapassar a limitação identificada na secção 2.3 e 2.4, é necessário fazer o levantamento dos termos e conceitos de negócio junto dos serviços e sistemas que foram construídos com base na documentação do departamento do produto, bem como as acções que são realizadas. O levantamento desta informação permite a sua utilização, adicionando semântica aos registos de *log*. Ter um log em que os valores dos dados tem associado a si o nome do domínio de negócio que representam, por exemplo: conjunto de vários vods - BVOD, vod comprado - PVOD, grelha televisiva - EPG, entre outros, permite caracterizar de forma explícita o que aquele valor é e simplificar o processamento analítico na caracterização desses valores. Outro aspecto relevante é a caracterização da acção realizada. É uma mais-valia ter presente no *log* o verbo que caracteriza a acção realizada, como por exemplo: comprar, alugar, ver, navegar. Um dos últimos problemas a resolver está relacionado com o envio dos *logs* para processamento analítico. Pretende-se que os serviços e sistemas consigam enviar os *logs* momentos após ao seu registo, não sendo necessário ficarem guardados localmente, evitando assim problemas de armazenamento nos servidores aplicacionais.

3 Conceitos base e trabalho relacionado

Este capítulo apresenta os conceitos base aplicados neste trabalho relacionados com o problema identificado no capítulo 2 relacionados com a especificação da estrutura de um novo *log*. Na secção 3.1 é apresentado o conceito base para o que se pretende definir. Na secção 3.2 serão apresentados os principais conceitos relacionados com a criação de uma ontologia. De seguida na secção 3.3 serão apresentados alguns trabalhos cuja a abordagem da definição de ontologias também foi seguida, e que utilizam ontologias que vão ser detalhadas na 3.2.

3.1 Eventos

O registo de logs com informação semântica do ecossistema aplicacional da NOS Inovação exige que se conheça o modelo de negócio implementado em cada um dos sistemas, e implica a adição de mecanismos de captura transversais a toda a aplicação, com baixo impacto no código aplicacional. Estes mecanismos quando adicionados, vão permitir enriquecer os processos de negócio permitindo obter conhecimento directamente dos sistemas que suportam a utilização das aplicações por parte dos clientes. A identificação, descrição, e análise de processos de negócio que ocorrem em sistemas aplicacionais é um trabalho que se insere na área de estudo de mineração de processos [1] e relevante na compreensão do correto funcionamento dos casos de utilização implementados nos sistemas. A mineração de processos consiste num conjunto de técnicas focadas na extração de informação do processo e não dos dados, com o objetivo principal de extrair uma descrição do modelo dos processos aplicacionais através da monitorização activa dos sistemas onde os processos estão implementados. Esta é uma das áreas de negócio para a qual também se pretende contribuir com este trabalho. A definição de um log semântico onde seja caracterizado o que é que aconteceu através de um verbo do negócio e onde também está presente a identificação das entidades do domínio aplicacional envolvidas, permite caracterizar a ocorrência de um caso de utilização aplicacional através do registo dos acontecimentos de cada um dos sistemas envolvidos que quando agregados permite caracterizar as ocorrências de casos de utilização.

A maior parte dos sistemas desenvolvidos na NOS Inovação que suportam as aplicações disponibilizadas aos clientes usam uma arquitetura orientada a eventos (Event Driven Arquitecture - EDA). A interação de um cliente numa aplicação gera um conjunto de eventos em um mais serviços. Se persistirmos essas ocorrências ao longo do tempo como entidades imutáveis com uma data e hora associada, consegue persistir a sequência de acções, que caracteriza o fluxo que o cliente realizou durante a sua utilização das aplicações cliente e o impacto que teve nos sistemas de *backend*. Esta metodologia é designada por *Event Sourcing* [2]. A utilização desta

metodologia no registo de eventos aplicacionais acrescenta um conjunto de características ao processo analítico que enriquecem o processo analítico, nomeadamente:

1. Registo de todas as alterações que aconteceram em cada um dos sistemas e modelos de negócio da organização.
2. Realizações de questões ao repositório de eventos, relacionando-os temporalmente e agregando-os pelas características que sejam relevantes para quem gere o produto.
3. Repetição de fluxos de negócio através da utilização dos eventos registados.
4. Recolha de padrões de fluxos de negócio. Ao identificar quais os eventos que ocorrem com sucesso para um cliente que faz uma compra, o processo analítico pode ser enriquecido com mecanismos de avaliação de padrões para os fluxos que são registados continuamente. Caso um fluxo que era de compra de conteúdos não acabe com sucesso a análise por padrões permite perceber o que está a fazer com que os clientes não comprem determinado conteúdo.

O processo de recolher e processar fluxos de eventos para realizar processamento analítico designa-se por *complex event processing* (CEP) [1, 3]. Para caracterizar a ocorrência de casos de utilização de negócio, a utilização de técnicas CEP é uma mais-valia no processamento analítico, permitindo a realização de operações de filtragem, agregação, composição, interrogações e identificar diferentes tipos de relações entre esses eventos, como por exemplo de correlação ou de causalidade [3]. Neste trabalho, uma acção que acontece num sistema, como por exemplo, a invocação de uma API HTTP por parte de outro serviço, é encarada como sendo um evento que ocorre nesse sistema, designado neste trabalho como sendo um evento aplicacional. Através das técnicas CEP no processamento analítico constroem-se eventos complexos que representam informação que queremos analisar, como por exemplo, se um caso de utilização ocorreu sem erros, ou quais os casos de utilização mais realizados por cliente.

Para poder registar eventos, será necessário primeiro definir o que é um evento no âmbito deste trabalho. Definir o que é um evento aplicacional, consiste na definição do domínio de conhecimento que representa uma ocorrência de um evento num serviço ou sistema. Esta caracterização passa por identificar quais os aspectos relevantes e como representa-los e como se relacionam entre si. Para esta tarefa existe uma área de estudo designada de ontologias.

3.2 Ontologias

A caracterização de um domínio de conhecimento pode ser feito com base em regras lógicas que descrevam os conceitos desse domínio de conhecimento [4]. Recorrer à lógica para a representação de conhecimento, requere que se identifique previamente o que se pretende representar do domínio a caracterizar. No âmbito deste trabalho pretende-se definir uma representação da ocorrência de um acontecimento aplicacional – evento aplicacional. Esta ocorrência deverá registar o maior conhecimento possível do contexto onde ocorreu e representar esse mesmo conhecimento numa estrutura homogénea e modular. Para descrever domínios de

conhecimento, existem famílias de linguagens formais na lógica para representar conhecimento, tais como, lógica proposicional, lógica de primeira ordem e lógica descritiva [4, 5]. Lógica proposicional não se aplica a resolução do nosso trabalho, uma vez que não permite descrever conceitos nem relaciona-los entre si. A lógica proposicional consiste apenas em exprimir factos sobre objectos simples onde uma afirmação apenas é verdade ou mentira. A lógica de primeira ordem é uma extensão à lógica proposicional uma vez que acrescenta um conjunto de operadores com os quais torna possível abranger um maior número de objectos na formulação de uma afirmação, tais como o símbolo ∀ de quantificação universal e o símbolo ∃ que quantifica a existência de um objecto. A lógica de primeira ordem acrescenta também outros símbolos que permitem a definição de afirmações, tais como, constantes, variáveis, predicados e funções. Em lógica de primeira ordem, o conhecimento é descrito na forma de predicados e regras [4], e esse não é o nosso objectivo. A descrição de um acontecimento num sistema aplicacional consiste em identificar um conjunto de conceitos e de terminologia com os quais se defina um vocabulário homogéneo por toda a organização no que se refere à produção de *logs* para processamento analítico. A lógica descritiva consiste nisso mesmo. A descrição do domínio de conhecimento recorrendo à lógica descritiva consiste na definição de dois conjuntos, o conjunto das terminologias, e da definição do *schema* designado por TBox e do conjunto das asserções composto por instâncias de objectos que são definidos pelas terminologias e *schemas* designado por ABox. Estes conjuntos podem ser descritos através da definição de uma ontologia.

Ontologias são importantes na modelação de um domínio de conhecimento [4], através da definição dos termos e seu significado e as relações entre si que representam esse domínio [5]. Dependendo do nível de detalhe do domínio em estudo é possível a reutilização de ontologia previamente definidas que abrangem os aspetos enumerados, ou até mesmo a criação de uma nova com base em várias ontologias. As ontologias disponibilizam um conjunto de características que permitem a representação de conhecimento de forma formal e por isso possível de ser interpretada por processos automáticos [4].

3.2.1 Definição

Uma ontologia define-se como sendo um método formal de especificar explicitamente a conceptualização do universo que se pretende modelar [5, 6, 7, 8, 9]. A conceptualização refere-se ao modelo abstrato do fenómeno que ocorre num determinado contexto caracterizado pela identificação da ocorrência dos conceitos relevantes que o caracterizam. Uma ontologia é uma especificação explícita uma vez que existe à priori a definição dos conceitos que existem relacionados com o domínio conceptual que se está a descrever e quais as restrições existentes associadas à utilização dos conceitos. A especificação formal dos conceitos e das relações permite que a partilha de conhecimento seja possível, sem introduzir outros significados a esses conceitos

nos diferentes sistemas onde esse domínio de conhecimento seja usado, pois representa um consenso sobre a representação de uma área de conhecimento [8, 10].

3.2.2 Características

As ontologias definem um conjunto de funcionalidades que permitem o processo de representação de conhecimento [4, 11]:

1. Vocabulário
2. Taxonomia
3. Partilha e reutilização de conhecimento

Uma ontologia é composta por um vocabulário especializado num domínio de conhecimento específico que define um conjunto de expressões lógicas que descrevem o que são os termos, como se relacionam uns com os outros e como é que não se podem relacionar [4]. Os termos que são definidos têm um significado único independentemente da língua em que são escritos e deverão ser entendidos por pessoas, e sistemas [4].

Taxonomia é a definição das relações hierárquicas das entidades identificadas num domínio, tais como, hierarquias de generalização entre conceitos, e a classificação [4].

A taxonomia e o vocabulário são a infraestrutura base na conceptualização do domínio a descrever [4] [12].

O objetivo da criação de uma ontologia baseia-se na partilha e reutilização de conhecimento definido pelo vocabulário e taxonomia [4]. A especificação de uma ontologia não deve por isso ficar restrita no meio onde foi desenvolvida, mas sim deverá ser partilhada entre pessoas para ser interpretada e analisada, mas também por sistemas e aplicações que abordem esse domínio de conhecimento, permitindo assim um conhecimento comum do domínio [13, 4]. A partilha de uma ontologia pode ser feita de diferentes formas dependendo da utilização a que estiver a si associada. No caso de a partilha ser entre aplicações e sistemas a partilha poderá ser feita em dois momentos [14]:

1. Tempo de desenho, através da partilha de componentes reutilizáveis na construção das aplicações e dos sistemas. Estes componentes teriam de ser cópias com o mesmo conteúdo permitindo o entendimento do vocabulário e da taxonomia associada de igual forma.
2. Em tempo de execução, onde os sistemas consumidores desse vocabulário e taxonomia teriam a capacidade de aceder à definição das ontologias através do acesso a serviços especializados.

O processo de desenho de uma ontologia é ele mesmo uma fonte de conhecimento e que pode ser utilizada para a definição de ontologias e que pode ser considerado para ser partilhado e reutilizado para a construção de ontologias desse domínio de conhecimento.

3.2.3 Princípios de desenho

Uma ontologia deve de ser suficientemente modular para poder ser reutilizada. Uma ontologia deverá ser um módulo com uma coerência interna forte com um número reduzido de interações com outros módulos. Alguns dos princípios de desenho recomendados são [8, 15, 13]:

1. Objectividade - Uma ontologia deve espelhar a intenção dos termos que define, as definições dos termos deverão ser objetivas.
2. Coerência - A definição de uma ontologia deve ser coerente com os termos que define em que as relações existentes entre os termos são coerentes com a sua definição.
3. Extensibilidade - A definição de uma ontologia não deverá ser fechada, e não deverá concretizar a existência de uma relação de 1 para 1 entre a sua definição e a instanciação num domínio aplicacional. Deverá existir pontos de extensibilidade para acrescentar definições de novos termos e conceitos e novas relações entre as definições.
4. Reutilização - A construção de ontologias de forma modular faz com que haja a reutilização de várias ontologias na definição de novas. A reutilização de ontologias existentes bem documentadas e validadas faz com que sejam criadas novas ontologias com algum nível de segurança de validade.
5. Separação de responsabilidades - A definição da estrutura de uma ontologia deverá estar separada do domínio de conhecimento dos sistemas. Esta separação permite que uma ontologia seja aplicada a diferentes domínios de conhecimento, como por exemplo a dos eventos.

3.2.4 Estrutura

Sendo uma ontologia uma representação abstrata de um domínio de conhecimento ela deve representar num formato formal capaz de ser interpretado por processamento automático. Esta característica só é possível se a definição da estrutura de uma ontologia recorrer a um conjunto de termos bem conhecidos cujo significado é interpretado da mesma forma por todos os processos automáticos que usam esses modelos. A estrutura de uma ontologia pode ser composta por um modelo de dados definidos com os conceitos:

1. Classes
2. Propriedades
3. Relações
4. Axiomas
5. Instâncias/Individuo

Classes representam a definição de conceitos que partilham o mesmo conjunto de características, que podem ou não ser conceitos físicos. Um local físico será uma característica física de um evento, enquanto o tempo não [16]. No mundo real existem diferentes tipos de eventos: eventos culturais, eventos naturais, eventos organizacionais, apesar de cada um destes tem as suas próprias

características todas elas partilham a definição de características comuns que definem o conceito base de evento, como por exemplo, local, data e hora, agente envolvido no evento.

Propriedades descrevem características de uma classe. Essas características podem ser definidas pela conceptualização de uma classe, ou pela instância de um valor estático. A definição de uma propriedade cujo seu valor é representado por uma instância do tipo de uma classe caracteriza a existência de uma relação entre os dois conceitos. Por exemplo, a ontologia de eventos em [17] relaciona o evento com um local, ou um evento está relacionado com um agente.

A definição de relações entre conceitos têm restrições associadas, asserções que têm de ser validadas à priori para que a relação quando instanciada seja verdadeira. As asserções de validação do conhecimento que se está a modelar podem ser feita a dois níveis. Pode ser feita a validação da estrutura definida e das relações entre as várias propriedades, a essa descrição de conceitos designa-se de TBox. A validação da construção correcta dos indivíduos dessa ontologia é feita por asserções através da instanciação de indivíduos designados por ABox. Comparativamente ao padrão de desenvolvimento de software orientado a objectos, pode-se comparar as asserções TBox como a definição de classes e propriedades, e as asserções ABox representam instâncias de uma definição de uma classe. O conceito mais específico que existe na definição de uma ontologia é a instância de um valor de um determinado tipo definido por uma classe que existe no domínio de conhecimento a modelar [16].

3.2.5 Formas de representação

O processamento de expressividade sobre um domínio representado por uma ontologia, tem como pré-requisito que a ontologia seja representada formalmente por uma das sintaxes que o Consortium do standard da web semântica conhece, entre as quais KIF, Ontolingua, Loom, SHOE, XOL, RDF OIL, DAML+OIL, OWL [4] e até mesmo UML [4, 18] sendo as mais comuns encontradas na bibliografia OWL [19] ou RDF [20].

RDF, *Resource Description Framework*, é uma framework composto por uma linguagem para representar informação de um domínio de conhecimento na forma de tripletos de informação composto por sujeito, predicado e objecto com o qual se pretende exprimir uma afirmação lógica sobre o domínio a modelar. Cada uma das partes do triplo é representado por um *Internationalized Resource Identifier* (IRI), literal, ou um nó sem valor. A representação do conhecimento de um domínio com RDF, concretiza-se na definição de grafos através da ligação entre vários triplos. O objectivo inicial da sua utilização seria o de representar metadata sobre conteúdos que existissem na internet, e por isso os componentes que formam esta framework e os elementos que definimos são identificados através de um URI. A representação dos conceitos descritos em RDF [21] podem ser serializados para um conjunto de formatos, tais como XML[3], Turtle[4] existindo também suporte

[3] https://www.w3.org/XML/
[4] https://www.w3.org/TR/turtle/

para a serialização para JSON-LD [22], um novo suporte de RDF para representação em JSON apoiado pelo próprio Consortium W3C do RDF [23].

Para extrair informação de um domínio de dados que é instanciado com RDF, é necessário conseguir interrogar o domínio de informação. Para suportar isso existe a linguagem SPARQL [24]. SPARQL é uma linguagem declarativa sintaticamente semelhante a SQL que disponibiliza um conjunto de operadores para realizar operações de filtragem, projecções, agregações sobre o formato de dados em grafo do RDF.

Web Ontology Language OWL é uma linguagem para expressar semântica num domínio sendo recomendada pela W3C. OWL É uma linguagem cuja sintaxe engloba a sintaxe definida em RDF(S) e acrescenta um outro conjunto de elementos sintáticos, que permitem exprimir relações semânticas que possam existir entre propriedades, classes ou indivíduos. O objectivo da sua utilização esta relacionado com a verificação de consistência de conhecimento de um domínio de informação. OWL, englobando a sintaxe de RDF(S) permite exprimir as classes e propriedades do domínio, quais os valores possíveis que as propriedades podem ter, quais as características das classes e adiciona ao que RDF(S) suporta a capacidade de exprimir semântica nas relações entre indivíduos e restrições semânticas à definição de tipos. OWL como apresentado na Figura 8, é uma infraestrutura composta por elementos sintáticos definidos nas linguagens de RDFS que por sua vez é um conjunto de é composto por elementos sintáticos da infraestrutura RDF.

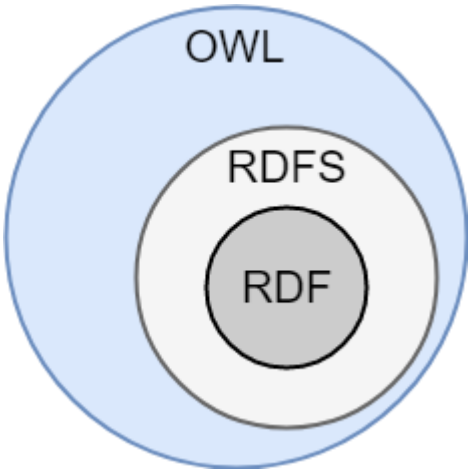

Figura 8 - Âmbito das várias sintaxes de ontologias

RDF é a infraestrutura que define uma linguagem mais simples composta por elementos sintáticos que permitem definir os triplos de dados[5]. Sintaticamente permite descrever relações entre dois recursos, como por exemplo, para exprimir a afirmação: O Vitor é Cliente da NOS, o seguinte grafo RDF apresentado na Figura 9 poderia ser produzido.

[5] https://www.w3.org/1999/02/22-rdf-syntax-ns#

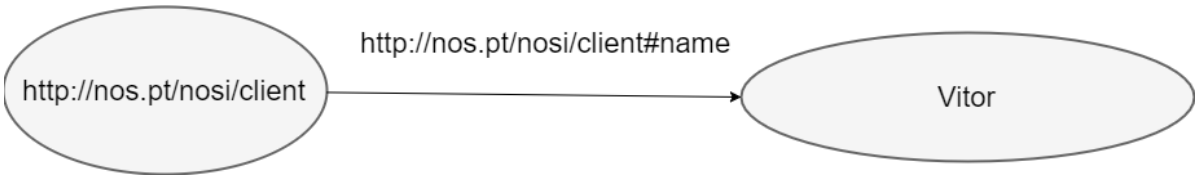

Figura 9 - Grafo RDF

O nó representado pelo IRI http://nos.pt/nosi/client tem uma relação com o texto vitor indicando o nome. Utilizando o formato RDF/XML a representação desta afirmação consiste no apresentado na Listagem 1.

```
<rdf:RDF xmlns:rdf="" xmlns:nosi="http://nos.pt/nosi/" >
    <rdf:Description rdf:about="http://nos.pt/nosi/client">
        <nosi:name>vitor paulino</nosi:name>
    </rdf:Description>
</rdf:RDF>
```

Listagem 1 - Exemplo RDF/XML

Outro formado possível para representar triplos de informação é JSON-LD [22, 23]. JSON Linked Data consiste numa representação do modelo de domínio numa estrutura JSON e foi considerado pelo grupo de trabalho do standard RDF como o formato a utilizar na representação com JSON [25]. Este formato, dentro da organização, é o formato eleito para transmitir dados entre aplicações, e portanto o conhecimento na organização sobre este formato é transversal a todas as equipas de desenvolvimento de software. A inserção de novos conceitos suportados sobre tecnologias, ou infraestruturas semelhantes às existentes vem permitir uma maior facilidade na aceitação na sua utilização. JSON-LD[6], ao contrário do JSON usado pelas aplicações, é um formato para descrever o meta modelo do domínio composto por elementos sintáticos próprios com os quais se define os tipos de dados do grafo de objectos bem como as suas relações e relações de significados entre conceitos. Outra das vantagens que existe na utilização de JSON-LD perante RDF/XML é a expressividade que existe em cada um dos formatos. Para descrever que vitor é um cliente usado RDF/XML é necessário a criação de mais elementos próprios da linguagem RDF enquanto que exprimir essa mesma informação a quantidade de texto próprio da sintaxe de JSON-LD é menor como é apresentado na Listagem 2.

[6] https://json-ld.org/spec/latest/json-ld/

```
{
  “@context”:{
      “nosi”: “http://nos.pt/nosi/”,
      “name”:”nosi:name”
  },
   "@type ": "nosi:client",
    “name” : “vitor”
}
```

Listagem 2 - Exemplo JSON-LD

Serializar para JSON-LD uma ontologia OWL torna necessário importar para o contexto do documento os espaços de nomes de RDF, RDFS e OWL, em JSON-LD isso faz-se recorrendo à palavra chave @context. Após importar para o contexto do documento esses espaços de nomes,

```
{
 "@context": {
   "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
   "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
   "xsd": "http://www.w3.org/2001/XMLSchema#",
    “owl”: ” http://www.w3.org/2002/07/owl”,
   “sse”:”http://nos.pt/nosi/analitics/structured-semantic-events”
 },
```

Listagem 3- Contexto JSON-LD dos espaços de nomes rdf rdfs e owl

como apresentado na Listagem 3, a utilização dos conceitos de cada uma das linguagens é possível, sendo da responsabilidade do processador de regras saber interpretar a estrutura JSON-LD.

A definição de uma ontologia com OWL no formato JSON-LD, consiste na utilização dos símbolos sintáticos desse espaço de nomes, com os quais é possível definir um documento JSON-LD recorrendo aos seus elementos sintáticos listados na página da especificação de JSON-LD [25].

Representar um evento aplicacional, consiste em definir as classes que representam os conceitos do que são importantes registar bem as propriedades desse domínio, definido assim o vocabulário. Em JSON-LD, a definição da classe Event consiste na definição de um objecto JSON como o exemplo apresentado na Listagem 4. A propriedade @id identifica o nome do tipo e a propriedade @type declara de que tipo é que o objecto do tipo definido em @id é. Com a utilização dos nomes definidos no contexto define-se o espaço de nomes de vocabulários será usado na definição da

```
{
 "@id": "sse#Event",
 "@type" : [ "owl#Class" ]
}
```

ontologia. Com a definição dos espaços de nome torna possível a utilização de conceitos tais como owl:class, rdf:range e rdf:domain. Desta forma torna-se possível definir as três principais componentes na definição de uma ontologia, classes, propriedades e ainda indivíduos.

```
{
  "@id" : " sse#hasAction",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ { "@id" : "sse#Event"   } ],
  "rdf#range" : [ {
    "@id" : "sse#EventActions"
  } ]
}
```

Listagem 5 - Definição de uma propriedade com sintaxe JSON-LD

```
{
  "@id" : "sse#browse",
  "@type":["owl#NamedIndividual", sse#EventActions" ]
}
```

Listagem 6 - Definição de um individuo de uma ontologia usando JSON-LD

UML é uma linguagem de modelação utilizada na análise e desenho de sistemas *object oriented programming* (OOP) definida como standard pelo *object management group* (OMG)[7]. UML tem uma abordagem ligeiramente diferente da utilizada nas restantes linguagens apresentadas, uma vez que modela as entidades e as suas relações através do desenho gráfico. Apesar de não existir nenhuma linguagem que traduza a representação gráfica em instâncias do modelo, existe um standard que define essa tradução designado por XMI[8]. A linguagem UML é uma linguagem de modelação com um conjunto de características que lhe confere a capacidade de ser usada para modelação de ontologias [4, 18], tais como:

(i) Permite a definição de classes, suas propriedades e relações com outras classes

(ii) ao contrário de linguagens de descrição lógica permite a existência de relações de 1 para muitos. Em UML é possível descrever o número máximo de relações que

[7] http://www.omg.org/spec/UML/
[8] http://www.omg.org/spec/XMI/

existem entre duas entidades, e em descrição lógica é apenas possível descrever que existe uma relação entre dois conceitos

(iii) Análise fácil por parte dos humanos

(iv) É a linguagem usada para modelar conceitos que se concretizam em componentes orientados a objectos, padrão de desenvolvimento usado na organização.

3.2.6 Construção

A construção de ontologias consiste num conjunto de princípios, processos iterativos, práticas e métodos e actividades para desenhar, construir e avaliar a ontologia que se pretende definir [4]. Atualmente não existe uma forma única de criar ontologias [16, 4], existem sim um conjunto de metodologias que definem um conjunto de passos, ou abordagens possíveis de seguir para a definição de uma ontologia. Em [4] são apresentadas metodologias de especificação de uma ontologia: (i) TOVE [26] (ii) Modelo Empresarial (iii) *Methontology* [27] (iv) KBSI IDEF5 [28] Através de uma análise comparativa destes métodos pretende-se identificar pontos em comum, com os quais se definam a metodologia que será seguida neste trabalho, tendo em conta o contexto organizacional onde este trabalho esta inserido.

A abordagem TOVE consiste num conjunto de pontos que começam por identificar os cenários de utilização, os termos, conceitos e relações existentes. Numa fase quase final consiste na especificação de axiomas relacionados com os termos e conceitos identificados e restrições associados a eles, terminando num processo de avaliação recorrendo a teoremas de completude.

A abordagem do modelo empresarial consiste em quatro passos: (i) identificar o âmbito da ontologia e o nível de formalidade necessário (ii) Identificar o âmbito de conceitos a representar na ontologia. (iii) Definição formal da ontologia (iv) avaliação do que foi definido no ponto (ii) e (iii). Esta abordagem, semelhante à anterior é composto por duas fases, uma mais informal, onde o levantamento dos requisitos da ontologia é realizada sem nenhum método formal associado. No caso da abordagem TOVE a fase do levantamento dos requisitos é algo tangível de se realizar através da identificação e análise dos principais casos de utilização. Semelhante a estes dois métodos mas num conjunto de passos mais detalhados temos a abordagem Methontology. Esta abordagem consiste num conjunto de passos que vão desde a especificação até à produção de documentação que descreva a ontologia criada, e consiste nos seguintes passos:

1. Especificação – Identificação do propósito da ontologia, casos de utilização, conceitos que se pretende representar as suas características e a granularidade associada.
2. Aquisição de conhecimento – Acontece por consequência do primeiro passo. À medida que se identificam os conceitos e suas características isso representa conhecimento sobre o domínio que se pretende representar

3. Conceptualização – após a recolha dos termos usados no domínio aplicacional, são definidos conceitos aplicacionais, instâncias desses conceitos relações entre conceitos através da definição de propriedades bem como verbos
4. Integração – Reutilização de conceitos de outras ontologias é vantajoso para permitir algum nível de homogeneização do domínio de conceitos.
5. Implementação – Representação formal da ontologia através de uma linguagem standard
6. Avaliação – Fase de validação e verificação da ontologia. Em [29] são apresentadas as linhas orientadoras de como realizar a avaliação.
7. Documentação – Agrupamento de todos os documentos produzidos nos restantes passos

A abordagem KBSI IDEF5 [28] é definida como sendo um conjunto de linhas de orientação versus um conjunto de passos formais fechados. Este método recomenda a construção a partir dos seguintes estágios:

5. Organizar o âmbito.
6. Recolha de dados.
7. Analise desses dados.
8. Desenvolvimento inicial da ontologia.
9. Finalização do desenvolvimento da ontologia

Este processo começa com a definição do objectivo da ontologia, que se identifique o contexto da criação da ontologia e quais os sistemas que vão participar para a sua definição. De seguida com o âmbito definido realiza-se a extração de todos os dados necessários, através da análise da execução dos sistemas e ainda entrevistas com especialistas desses sistemas. Com todos os dados recolhidos é necessário analisa-los e a partir deles definir um protótipo da ontologia. Este protótipo é composto por conceitos relações e propriedades iniciais identificados da extração realizada dos dados. Num processo iterativo, os conceitos propriedades e relações vão sendo refinadas até chegar à definição da ontologia final.

Todas estas metodologias para a construção de ontologias apesar de nomes diferentes, ou tenham uma maior granularidade no processo, ou se cruzem com outras formas de representar conhecimento, todas elas são semelhantes, todas definem a necessidade de identificar o âmbito, conceitos e vocabulário e definir como é que esses conceitos se relacionam entre si. A construção no âmbito organizacional, como o da NOS Inovação, requere que se estude os sistemas, e processos existentes para que a ontologia suporte a actualidade, mas que suporte também o crescimento dos domínios aplicacionais e as acções possíveis que esses sistemas realizam.

Estas metodologias, quer seja através de um conjunto de passos formais, quer seja na forma de linhas orientadoras definem abordagens que identificam necessidades comuns. Das diferentes necessidades que cada um dos métodos identifica na forma de passos a realizar, todos eles caracterizam os passos listados em [16]. As considerações apresentadas são consideradas fundamentais nos passos que são descritos e que estão presentes nas abordagens apresentadas:

1. A solução depende da aplicabilidade da ontologia
2. A definição de uma ontologia é um processo iterativo
3. Os conceitos de uma ontologia deverão estar perto dos objectos que serão instanciados e das relações que existem entre eles. Ao exprimir uma ontologia vão existir substantivos que representam os objectos e/ou verbo que expressam as relações no domínio a ser descrito pela ontologia.

Os passos que compõem a construção de uma ontologia apresentado em [16] são comuns às abordagens apresentadas e consistem em:

1. Definir o âmbito do domínio em estudo
2. Reutilização de ontologias existentes
3. Enumerar os termos mais relevantes da ontologia
4. Definir os conceitos, as relações entre eles, propriedades e axiomas – Asserções TBox
5. Definir instâncias dos conceitos definidas – Asserções ABox

A evolução e manutenção de ontologias é um aspecto a ter em conta, uma vez que domínios de informação de diferentes naturezas estão constantemente a evoluir. Este aspecto faz que exista cada vez mais o interesse e a necessidade na criação de ontologias modulares [30]. Embora este não seja um aspecto detalhado no processo da criação da ontologia é importante que a definição de conceitos seja modular para que possa ser reutilizável, tanto na própria ontologia como em outras.

3.2.6.1 Definição do âmbito de uma ontologia

O primeiro passo para a criação de uma ontologia consiste em identificar qual o âmbito do domínio de conhecimento que se pretende abordar. Para este levantamento deverão ser respondidas algumas questões cujas respostas ajudam a definir as fronteiras dos conceitos a criar e o nível de relações que existem entre elas, quer seja hierárquico quer seja de composição. Algumas dessas questões são:

1. Qual o domínio de conhecimento que queremos abordar?
2. A ontologia vai ser usada com que finalidade?
3. Que tipo de questões vão ser colocadas sobre o domínio definido por essa ontologia?
4. Quem é que vai usar e manter a atualidade da ontologia.

3.2.6.2 Reutilização de ontologias existentes

A reutilização de ontologias é um aspecto a considerar porque permite a reutilização de conceitos já criados num domínio de conhecimento que abrange na totalidade ou apenas parcial a ontologia que se está a criar. Outro aspeto é a partilha de informação. Se um dos requisitos da ontologia é suportar sistemas que irão partilhar informação de um domínio de conhecimento comum entre várias organizações, a criação de uma especificação de como é que a informação está estruturada é uma mais-valia, facilitando a integração entre os vários sistemas e organizações. A reutilização

de ontologia permite também uma redução de recursos gastos e evitar a redefinição de conceitos já existentes. Um exemplo disso são as organizações que disponibilizam conteúdos EPG e VOD, como é o caso da NOS. A partilha de como é que o domínio está organizado e como é que os conceitos se relacionam entre si, facilita a integração de informação de diferentes fontes, quer seja vindo de outras operadoras de TV Cabo.

3.2.6.3 Enumeração dos termos mais relevantes da ontologia

No processo de caracterizar o domínio de conhecimento, é necessário identificar as palavras-chave e conhecer os principais termos e conceitos presentes no domínio. Este levantamento não consiste só na sua identificação, mas também é necessário fazer a sua caracterização. Alguns desses conceitos representam entidades que são descritas por um conjunto de propriedades e que por sua vez se relacionam com outras entidades. Este levantamento vai permitir a execução do próximo passo, a definição das classes, das suas propriedades, relações entre si e a definição de axiomas do modelo.

3.2.6.4 Definição das classes e hierarquia, propriedades e axiomas

A definição das classes, das suas propriedades e das relações com outras entidades e axiomas é a execução da formalização num formato descritivo formal que pode ser feito de três formas [16]:

1. *Top-down.*
2. *Bottom-up.*
3. Combinação entre as duas anteriores.

Uma abordagem *top-down* consiste na definição de conceitos do domínio do mais geral para o mais particular. Um evento aplicacional que se pretende registar ocorre num servidor, onde está instalado um serviço, que é composto por um conjunto de componentes. Outro exemplo é a definição do utilizador aplicacional. Um utilizador aplicacional, pode ser identificado apenas pela conta de cliente que identifica o contrato do cliente para com a empresa proprietária da aplicação, um perfil de consumo, e por fim temos o utilizador que é identificado univocamente pelo seu login.

A abordagem *bottom-up* consiste na identificação imediata de conceitos específicos de áreas do domínio, que agrupam características comuns levando à criação de uma classe base desses conceitos. Como por exemplo, os substantivos que caracterizam o sujeito da acção que o evento caracteriza, e os argumentos que caracterizam a rotina do serviço que foi executada são entidades que podem ser descritas na sua base da mesma forma, têm um identificador único e um valor.

Por fim, temos a utilização em simultâneo destas duas técnicas para a especificação da ontologia.

3.2.6.5 Definir instâncias de objectos das classes definidas

O último passo na definição de uma ontologia é a instanciação de conceitos que utilizem as classes definidas na ontologia definida. Essas instâncias designam-se por indivíduos. A definição de indivíduos tem aplicabilidade na definição de uma ontologia, quando se conhece todo o domínio

de valores possíveis de um conceito. Como exemplo neste trabalho podemos encontrar a definição de indivíduos relacionado com:,

1. A definição de um conjunto de indivíduos que estabeleça o âmbito do tipo de aplicações é que existem na NOS Inovação,
2. Conceitos de negócio presentes no ecossistema da organização.Com uma abordagem *bottom-up* faz-se o levantamento de todos os conceitos que definem domínios aplicacionais existentes, e cria-se uma entidade com a qual os consiga representar. A criação destes indivíduos permite uma caracterização semântica dos domínios aplicacionais das entidades presentes no evento.
3. Verbos que representam as acções que ocorrem nos serviços e sistemas.

3.2.7 Ontologias modulares

O conceito base de ontologias modulares assemelha-se ao do existente no desenvolvimento de software modular. O conceito consiste em decompor uma ontologia monolítica em componentes mais pequenos com o objetivo de:

1. Facilitar reutilização de conhecimento por diferentes aplicações.
2. Maior facilidade na construção, manter e substituir.
3. Permite uma distribuição de módulos para a construção de ontologias por diferentes locais e diferentes áreas de estudo.
4. Possibilitar uma consulta e gestão mais eficaz de ontologias através dos seus módulos [12]. A modularização de ontologias pode acontecer de duas formas:
    a. Na criação de uma nova ontologia baseada em outras ontologias, cria-se um módulo por cada ontologia referenciada [31] ou
    b. Particionamento de uma ontologia existente com o objectivo de extrair módulos [6]. O processo de criação de módulos requere que hajam critérios para que isso aconteça. Os critérios para a modularização de uma ontologia pretendem que a modularização aconteça, sem que haja perca de informação para os módulos criados, critérios como o encapsulamento a coesão, redundância entre módulos [10].

3.2.8 Ontologias existentes

As ontologias podem-se caracterizar como sendo de alto nível, ou especificas de domínio. Uma ontologia de alto nível, contém a definição de conceitos transversais a quaisquer domínios de conhecimento, enquanto uma ontologia de domínio, define os conceitos específicos de domínio de conhecimento que se pretende modelar [13, 8]. O âmbito da definição da ontologia deste trabalho está inserido num contexto organizacional, e como tal, a definição da ontologia será orientada a conceitos organizacionais existentes, mas fundamentados por conceitos já existentes em ontologias de alto nível ou em ontologias de domínio. A abordagem deste trabalho consiste

na definição de uma ontologia com a qual se consiga caracterizar através de conceitos existentes na organização e no sistema actual de extração de dados, eventos que ocorram nos sistemas da NOS Inovação. No domínio do conhecimento dos eventos já existem ontologias definidas, que se caracterizam por ser ontologias de alto nível, e também ontologias específicas de domínios de conhecimento.

Da pesquisa feita por ontologias que envolvem conceitos relacionados com eventos, surgiram as seguintes ontologias:

1. *ABC Ontology* [32]
2. *Event Ontology* [33]
3. DOLCE+DnS Ultralite (DUL) [34]
4. Event Model-F [35]
5. IPTC. EventML [36]
6. LODE [37]
7. OpenCyc [39]

Estas ontologias algumas são de alto nível cujos conceitos são descritos como possíveis de ser reutilizados para descrever eventos, tais como DOLCE+DnS Ultralite, Event Model-F que é uma especialização da DOLCE+DnS Ultralite ou OpenCyc. Estas ontologias sendo de alto nível, não são focadas na representação de eventos, o que elas permitem, devido ao extenso vocabulário que as define, é a descrição de eventos com várias componentes, como por exemplo a componente temporal, de actores envolvidos, local onde o evento ocorreu, o que foi produzido nesse evento. Para a construção da ontologia deste trabalho, pretende-se a reutilização de conceitos focados em eventos. Num contexto organizacional como a NOS Inovação, pretende-se que as pessoas que utilizem esta ontologia se foquem nos conceitos relevantes a extrair e a analisar. Embora estas ontologias sejam ricas em conceitos, a sua utilização em excesso podem introduzir confusão. O que se pretende destas ontologias é captar os conceitos fundamentais para a definição da ontologia deste trabalho. Da listagem de ontologias apresentadas focou-se o estudo nas ontologias apresentadas na Tabela 2 por duas razões. São ontologias focadas no conceito do evento como é o caso da *Event Ontolpgy e LODE* e porque estão associadas a casos reais apresentados na secção 3.3 como é o caso da *Model F ontology*.

Tabela 2 - Lista de ontologias de eventos

Nome	**Tipo**	**Descrição**
LODE	Ontologia superior	Eventos são vistos como objectos ligados entre si
Ontologia de Evento	Ontologia de domínio	Ontologia inicialmente criada para descreve música digital

Event Model F	Ontologia superior	Descrever sistemas que funcionam com base em eventos

A ontologia LODE designa-se por ser a descrição minimalista que contém os atributos essenciais para descrever um evento [37]. O objetivo desta ontologia é a de permitir a interoperabilidade na definição dos factos de um evento. Estes aspectos são caracterizados pela regra dos 4 W's:

1. O que aconteceu - *What Happened.*
2. Onde é que aconteceu - *Where did it happen.*
3. Quando é que aconteceu -*When did it happen.*
4. Quem esteve envolvido - *Who was involved.*

Uma das razões que mereceu um estudo detalhado a esta ontologia neste trabalho deve-se ao facto de que o conceito principal do vocabulário desta ontologia é Event. Event é uma classe que define algo que aconteceu, limitado no tempo e no espaço num ambiente real ou fictício sobre o qual se pretenda realizar asserções que relacionem pessoas, lugares, coisas a eventos. Estes conceitos na ontologia estão disponíveis através das suas propriedades listadas na Tabela 3

Tabela 3 - Vocabulário da ontologia LODE

Nome	**Descrição**
atPlace	valor do local onde aconteceu o evento, este pode ser uma entidade ou um conjunto de entidades que caracterizem o local
atTime	Informação abstrata da noção temporal que serve de aproximação ao momento em que o evento correu
Circa	Valor da data e hora a que o evento ocorreu.
Illustrate	Descrição do evento de forma ilustrativa, figurativa
inSpace	Descrição espacial onde ocorreu o evento. Caso este valor seja demasiado vago o seu significado não é o de indicar que o evento aconteceu em todos os locais dessa região, mas sim que ocorreu algures nessa região
Involved	Descrição de algo que esteve envolvido na ocorrência do evento, não de forma causal, mas sim como entidade que fez parte da ocorrência do evento
involvedAgent	Descrição da participação de um agente no evento, não de forma causal, mas sim como entidade que fez parte da ocorrência do evento. Um agente pode ser uma pessoa, organização, processo automático etc.

A ontologia LODE define conceitos que são fundamentais para o processo CEP que se pretende construir na organização, conceitos esse também identificados por David Luckham [3]. A regra dos 4W aplicada aos eventos aplicacionais recolhidos dos sistemas vai permitir um processamento CEP focado a responder questões de negócio como também de monitorização. O conceito de saber o que aconteceu descrito pela propriedade Illustrate na extracção de eventos aplicacionais não é aplicável, uma vez que não é possível gerar um objeto media (imagem ou vídeo), mas o relevante que se pretende aproveitar deste conceito é a existência de um conceito que permite caracterizar o que aconteceu. No caso de um acontecimento fará mais sentido descrever o que aconteceu com um verbo por exemplo. Saber quem esteve envolvido pela definição de Involved ou involvedAgent são semelhantes ao que actualmente é extraido do processo de recolha, nomeadamente identificação do cliente, identificação do equipamento que o cliente usou. Com informação de onde é que aconteceu e quando é que aconteceu permite ao processamento CEP identificar sob que condições é que o evento aplicacional aconteceu, nomeadamente se for descrito pela informação de que servidor, aplicação ou versão da aplicação é que gerou esse evento. A noção temporal é um conceito que caracteriza a ocorrência de eventos e que está presente em maior parte das ontologias listadas no início desta secção. Ter uma noção temporal de quando e que um evento ocorreu é importante para poder correlacionar essa ocorrência com outras que ocorram no mesmo intervalo de tempo, criando agregações por cliente, por sistema ou outros conceitos relevantes.

A ontologia de evento apresentada na Figura 10, é relevante porque centra-se na definição de evento [17]. Esta ontologia acabou por não ficar com nenhum conceito diretamente relacionado com o universo da música. A definição de evento com esta ontologia consiste numa classificação arbitrária de espaço e tempo por um agente cognitivo. Um evento caracteriza-se por um conjunto de agentes designados de participantes, agentes passivos, agentes que não estão presentes na ocorrência do evento mas que não influenciam a sua ocorrência, produtos e uma localização num determinado espaço e tempo.

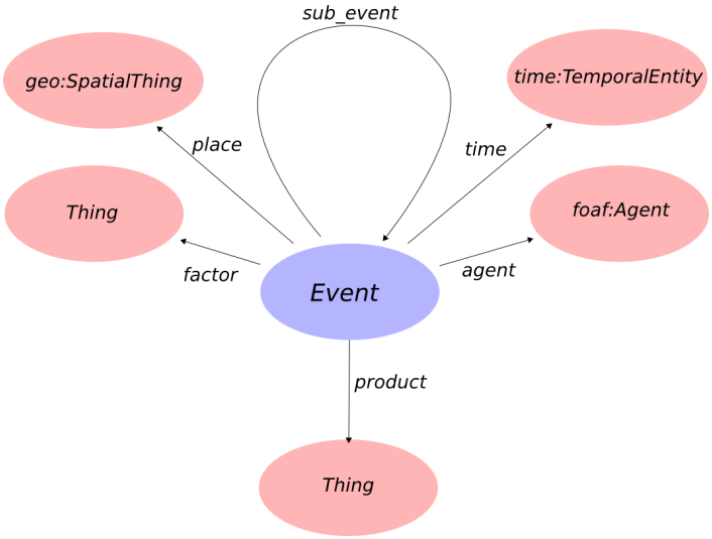

Figura 10 – *Event Ontology* adaptado de [17]

Como propriedades esta ontologia define: (i) Produto, (ii) Factor, (iii) Agente, (iv) Local e (v) Tempo. O produto representa algo que seja produzido no evento. A propriedade factor identifica participantes num evento não havendo distinção entre um participante que foi influenciado pelo evento, ou de participante que participou ativamente nesse evento. O espaço onde o evento ocorre é representado por um objecto na propriedade *place* do tipo *SpatialThing* definida na ontologia de posicionamento geográfico [38]. A relação que existe entre eventos e o tempo é presentado pela propriedade time, esta propriedade deriva da definição de ontologias específicas para a modelação do domínio temporal, como por exemplo a ontologia do tempo [41]. Um evento é definido pela agregação de outros eventos e essa relação também tem de se conseguir modelar de forma formal. A ontologia evento disponibiliza essa funcionalidade através da relação entre dois eventos designada por sub-evento. Esta ontologia no entanto apenas permite relações simples de primeiro nível entre eventos, não sendo capaz de modelar relações merológicas mais complexas. Esta ontologia apresenta conceitos que todos eles são válidos no contexto deste trabalho, excepto a localização ser representada por uma ontologia de um local georeferenciado. A relação temporal é algo que no próprio conceito de evento é mandatório associar. Saber quando aconteceu, quando abaou, quanto tempo durou são representações relevantes pois permitem ao processo CEP correlacionar eventos simples na criação de eventos complexos, como por exemplo, Identificar todos os casos de utilização que começaram entre um período do dia, ou medir o tempo médio de execução de um caso de utilização. Saber o quê e quem esteve envolvido são características também relevantes, pois mapeam para o nosso trabalho para a identificação do domínio aplicacional envolvido e qual a entidade de domínio e quem foi participante envolvido, ou em que *software* o evento ocorreu.

A ontologia Model F é um modelo formal de eventos desenhado para capturar e representar experiências do quotidiano humano. Esta ontologia disponibiliza no seu vocabulário suporte para representar tempo, espaço, objectos e pessoas e ainda informação meteorológica. Esta ontologia

disponibiliza ainda capacidades para criar eventos complexos, modelar causalidade de eventos e correlação, características essas requisitos na definição de eventos complexos em CEP. Model F é uma ontologia modular composta por outras ontologias específicas dos objectos podendo ser extendida para a definição de ontologias de domínio [35]. A Figura 11 apresenta uma visão global desta ontologia onde são apresentados todos os conceitos e suas relações.

A ontologia *Model F* foi desenvolvida com base noutras duas ontologias já previamente combinadas, DOLCE+DnS ultralite [35]. Esta combinação permitiu a aglomeração numa só ontologia um conjunto de conceitos que permitem descrever situações de forma modular, relações entre conceitos tais como localização e temporal. Embora sejam ontologias de alto nivel, estas são ontologias que separam os conceitos daquilo que é um evento de um objecto abstrato. Um evento é uma entidade que existe e evolui no seu estado limitado no tempo, enquanto que um objecto limita-se a existir [35].

Apesar deste aglomerado de conceitos e relações, a ontologia Model F, foi desenhada para fazer cumprir um conjunto de requisitos funcionais e não funcionais que estivessem já presentes em modelos de eventos definidos noutras ontologias de domínio de modelação de eventos. As ontologias que motivaram a criação de model F são: Eventory uma ontologia de jornalismo [40], modelação de eventos [33], EventML [41] para descrever noticias, ontologia de modelação de eventos de vídeo e ainda modelação de aplicações multimédia orientada a eventos [42, 43]. Os requisitos funcionais que esta ontologia implementou são:

1. Definição de eventos com objectos para caracterizar entidades que existem relacionados com o evento.
2. Caracterização temporal e espacial de um evento.
3. Ser possível definir relações de composição, causalidade e correlação e entre eventos através da sua estrutura.
4. Anotação de eventos permitindo documentar os eventos ou os seus objectos constituintes.
5. Permitir a interpretação de um evento por diferentes pontos de vista.

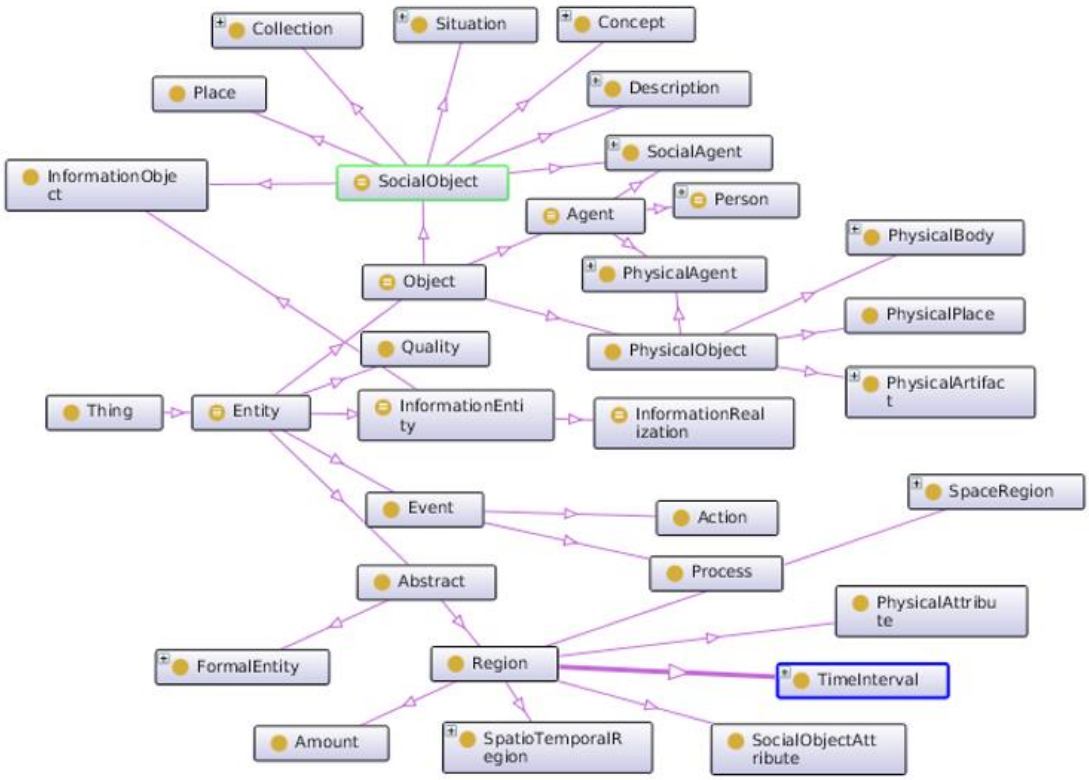

Figura 11 - ontologia Model F [35]

Estes são os pontos que a ontologia Model F implementou que definem o que uma ontologia definida com este modelo deverá suportar. Outro grupo de requisitos que também foram definidos são requisitos não funcionais, que especificam como é que uma ontologia deve ser definida. Estes requisitos são: (i) extensibilidade para permitir que a ontologia evolua ao ritmo do domínio que modela (ii) Modularidade permite a utilização das diferentes partes da ontologia em separado bem como reutilização dessas mesmas partes para a definição de outras ontologias.

Model F define no seu vocabulário um conjunto de classes fundamentais para atingir os requisitos funcionais definidos. Estas classes são:

1. Event como sendo uma entidade que existe algures no tempo.
2. A classe Object que representa uma entidade que existe num espaço, quer seja físico ou não, quer seja uma entidade viva ou apenas coisas abstratas.
3. *Quality*. A qualidade é outra classe desta ontologia e é uma característica de um objecto ou evento.
4. Uma qualidade tem um valor associado que existe no espaço. Este valor é um *Abstract*. Exemplos disso podem ser um intervalo de tempo, uma região válida no tempo e no espaço ou apenas uma região de um determinado espaço.

O conjunto de requisitos funcionais são resolvidos pela aplicabilidade de padrões de desenho específicos para cada uma das necessidades. Os padrões de desenho usados na definição desta

ontologia são os apresentados na Tabela 4 e permitem, descrever formalmente os participantes, as relações entre os objectos e eventos por causalidade ou por correlação, documentar as entidades das ontologia e criar interpretações da ocorrência de um evento.

Tabela 4 - Padrões de desenho da ontologia Model F [35]

Padrão de de desenho	Descrição
Participação	Exprimir a participação de de objectos em eventos
Mereologia	Exprimir a relatividade de relações temporais e espaciais entre eventos.
Causalidade	Exprimir a causa da ocorrência de eventos derivada de outros eventos
Correlação	Relacionar eventos quando estes são independentes entre si, ou seja, não são a causa um do outro
Documentação	Um evento ou objecto deverá ser documentado ou descrito através de uma entidade
Interpretação	O mesmo evento, quando observado por diferentes observadores pode ser interpretado de diferentes formas, criando diferentes modelos da aplicação dos padrões de desenho desta tabela.

Com esta ontologia superior, tirando partido da utilização de diferentes padrões de desenho para modelar cada um dos requisitos funcionais é possível criar representações de acontecimentos do mundo real e modelar formalmente as diferentes relações que existem entre os vários participantes presentes nesse evento bem como diferentes interpretações do mesmo evento, possibilitando também a criação de relações complexas entre eventos.

Na investigação das ontologias existentes para a modelação de eventos verifica-se que cada uma destas ontologias é definida por conceitos a diferentes níveis. As ontologias de alto nível definem conceitos transversais a domínios aplicacionais mas distinguindo no entanto o conceito de evento diferente de definições genéricas. As ontologias de domínio aplicacional por outro lado definem os seus conceitos com expressividade associada ao domínio.

Independentemente do nível de abstracção das ontologias todas estas detalhadas possuem conceitos que são fundamentais para a formalização de uma ontologia para modelar eventos, nomeadamente, atributos que guardam informação temporal, local onde ocorreu o evento, actores ou agentes envolvidos ou responsáveis pela ocorrência do evento. Outro aspecto que se pode observar das diferentes ontologias estudadas neste trabalho é que todas elas modelam uma estrutura para responder a um conjunto de questões, como se de uma investigação policial se tratasse, tais como:

- O que aconteceu?

- Quando é que aconteceu?
- Onde é que aconteceu?
- Quem foram os envolvidos?
- Porque é que aconteceu?
- Como é que aconteceu?

Para cada uma destas questões cada ontologia definiu um conjunto de atributos, , que se relacionam entre si. Nas ontologias apresentadas o tempo é modelado, de diferentes formas sendo a mais significante a representa pela ontologia do tempo [41]. A ontologia do tempo define conceitos relevantes para este trabalho, tais como o que é um Instante temporal, um intervalo de tempo e propriedades que expressam semântica relevante, nomeadamente instantes temporais que marcam o inicio e o fim de uma entidade temporal. No que se refere ao local, todas as ontologias relacionam o evento com um local onde ocorreu, no caso da ontologia do evento, o local é definido como um local identificado num espaço físico pela ontologia de *geospatial*[9]. Das ontologias previamente listadas o local do evento remete em todos os casos para um local físico. Neste trabalho isso não acontece. Tratando-se de registar dados de serviços ou sistemas que estão em execução em servidores, interligados por redes de comunicação que permitem a que sistemas comuniquem entre si, o conceito de localização tem outro significado, ou seja, consiste em saber em que servidor e/ou aplicação e se possível o nome do componente da aplicação bem como o nome da rotina. Caracterizando o Local desta forma, existe uma diferente clara do que é um local físico, do que é um local na ocorrência de um evento aplicacional. A relação entre eventos, como a causalidade é modelada de forma diferente pelas ontologias apresentadas. A ontologia LODE não apresenta essa capacidade, a ontologia do evento apresenta uma relação de primeiro Grau entre dois eventos, a ontologia *Model F* é a mais completa a modelar relações entre eventos, suportando os padrões de mereologia, causalidade e correlação. Das ontologias estudadas, a Model F aparenta ser aquela que abrange um maior número de conceitos utilizados num processo CEP. A causalidade neste trabalho consiste em saber quem causou a ocorrência do evento aplicacional que está a ser registado. Quem causou abrange diferentes conceitos, tais como, identificar o cliente que executou, ou sistema que originou a execução ou a partir de que equipamento é que causou a ocorrência do evento.

A utilização de ontologias na definição formal de eventos complexos pode ser vantajoso para o processo CEP uma vez que enriquece o processo com o conceito de dar semântica a um evento, podendo trazer mais qualidade às regras de filtragem, agregação e correspondência de padrões.

[9] https://www.w3.org/2005/Incubator/geo/XGR-geo-ont-20071023/

3.3 Processamento semântico de eventos complexos

Processamento semântico de eventos complexos (SCEP) é uma área de estudo que aborda a utilização de tecnologias da web semântica no processamento de eventos complexos [44, 45]. A utilização de ontologias para definir os conceitos a aplicar num processo CEP pode ser vantajosa uma vez que enriquece o processo com semântica, podendo trazer mais qualidade aos dados que são recolhidos na sua expressividade, às regras de filtragem, agregação e correspondência de padrões. Processos de construção de eventos complexos, têm na sua definição, conceitos que também estão presentes na estruturação de eventos através de ontologias, quer seja na sua estrutura com a presença de um atributo, quer seja através das suas regras de filtragem, composição e agregação.

Em CEP são definidos mecanismos para criar, relacionar, agregar eventos, nomeadamente através da criação de agentes de processamento de eventos (*Event processing agente* EPA) [3] que executam regras para detectar padrões e assim tomar decisões sobre o que fazer de seguida. Nas ontologias, a execução de regras para expressar conhecimento ou para validar a coerência do domínio também é algo possível de se executar, recorrendo a linguagens de interrogação como por exemplo SPARQL. Este é um ponto que ambas as áreas têm em comum, ou seja, definem e sugerem ferramentas para a extração de conhecimento. No que se refere à especificação formal dos conceitos e definição da expressividade que os conceitos possuem e de como se relacionam entre si, só as ontologias possuem essa capacidade através da definição de axiomas. A semântica dos eventos deverá ser definida de forma mais exacta possível, recorrendo a tecnologias de web semântica.

A utilização de tecnologias tais como RDF e OWL com CEP surgiram num conjunto de trabalhos [13, 46, 47, 48] com o objectivo de modelar e exprimir formalmente os domínios envolvidos.

O trabalho de Zhu, Bakshi, Prasanna and Gomadam [46] consistiu no desenvolvimento de uma ontologia OWL para representar os eventos que ocorriam num reservatório de água. Esta ontologia define os tipos de events que podem ocorrer num reservatório bem como as propriedades que os constituem. Esta ontologia define ainda as relações existentes entre os diferentes tipos de eventos, relações sequenciais, relações organizacionais, e relações causais. Neste trabalho os autores definiram ainda um conjunto de *regras* semânticas com *semantic rules language* SWRL que mapeando relações de causalidade identificavam a ocorrência de eventos aos quais tinham de estar atentos causados por outros.

A área de estudo de sensores RFID, é um domínio de conhecimento rico em eventos e Liu e Guan [48] especificaram uma ontologia de domínio em OWL designada por *Card* para formalizar o domínio de informação de viajar em transportes públicos, onde um conjunto de sensores RFID geram eventos de entrada e saída dos meios de transporte públicos, eventos esses que eram guardados numa base de dados relacional. Esta ontologia define os tipos de cartão disponíveis e as suas propriedades. Era objectivo deste trabalho, conseguir realizar interrogações à base de

dados com base na ontologia por forma a conseguir calcular descontos entre dois eventos (de entrada e saída) que aconteciam para um cartão. Uma vez que não é possível aplicar questões em SQL sobre uma ontologia, a abordagem deles, consistiu em implementar um interpretador de SQL próprio para ontologias.

A utilização de ontologias para formalizar a especificação de eventos em CEP foi a base do trabalho europeu WeKnowIt [13]. Este projecto teve como objectivo formalizar a estrutura dos dados que diferentes sistemas de diferentes departamentos de controlo de emergências partilhavam, bem como os eventos que eram gerados e enviados para os restantes serviços, tais como, serviços da polícia, departamento de bombeiros ou gabinetes de protecção civil.

Uma das ontologias usadas neste projecto foi Model F. Esta ontologia, já descrita na secção 3.2.8, foi usada neste projecto para descrever os eventos que eram gerados, bem como as múltiplas relações entre eles, possibilitando assim a comunicação entre as diferentes entidades com uma estrutura formalizada e reconhecida pelos sistemas de todas as entidades envolvidas.

Num domínio de conhecimento diferente, a aplicação de ontologias para enriquecer o processo CEP foi tema no trabalho de Monica L. Nogueira. Este trabalho tinha como objetivo a deteção de surtos de doenças alimentares [47]. Neste trabalho, os dados que eram processados tinham como origem texto não formatado que quando processado despoletava a execução de agentes que através de um processo de regras inferia o que tinha acontecido com base no texto recebido. O resultado desse processo despoletava eventos a notificar o que supostamente terá acontecido. Através da utilização de ontologias de comida, esses eventos eram relacionados sendo geradas novas evidências de novos tipos de surtos de doenças relacionadas com alimentos.

As abordagens aqui apresentadas usam características das ontologias para enriquecer o processamento CEP, tais como, definição do vocabulário, taxonomia, reutilização, e extensibilidade [8]. Nexte contexto, verifica-se que a integração de tecnologias da web semântica em CEP é uma área de estudo com interesse, com a detecção de padrões de eventos de forma rápida [49] e com mais semântica [50].

4 Solução proposta - Ontologia

Este capítulo apresenta a definição da ontologia deste trabalho. Esta ontologia tem como objectivo, representar a ocorrência de eventos aplicacionais. Esta ontologia entra para a categoria de ontologia de domínio, ou seja, o seu domínio de conhecimento é específico para a caraterização de ocorrências de acções em serviços ou sistemas. Na secção 4.1 é definida a abordagem seguida para a definição da ontologia. Na secção 4.2 é feita um estudo ao log actual, com o objectivo de extrair o que de relevante ele tem, nomeadamente o seu conteúdo. Nesta secção, é ainda analisada a utilização do *log* no processamento analítico, o que é relevante e é usado, e o que não é usado mas que é uma mais-valia para a geração de conhecimento do negócio da NOS. Na secção 4.2 são apresentados os serviços e sistemas que serão utilizados para o registo de eventos. Nesta secção, pretende-se apresentar os diferentes conceitos de domínio aplicacional existente que vão ser mais tarde incorporados na ontologia como indivíduos de uma classe. Na secção 0 são apresentados os principais casos de utilização usados atualmente no processamento analítico. A descrição destes casos de utilização tem como objectivo, identificar as operações que ocorrem entre os sistemas extraindo dessa análise informação relevante para a definição da acção da ontologia. Semelhante à definição dos indivíduos dos domínios aplicacionais, pretende-se nesta secção identificar os verbos de acções que ocorrem nos sistemas e com estes definir um conjunto de indivíduos de uma classe que represente o nome das acções possíveis que um evento pode ter. Na secção 4.5 com o resultado da análise das secções anteriores será apresentada a definição da ontologia deste trabalho.

4.1 Abordagem

A solução proposta neste trabalho segue uma abordagem *bottom-up* [7, 17, 42] e foi organizada em três fases: (i) Estudo dos principais casos de utilização disponibilizados ao cliente, identificar os domínios de conhecimento dos *logs* existentes, do processamento analítico actual, do ecossistema de serviços e sistemas. Esta informação servirá para definir a base de conhecimento da ontologia, ou seja, a definição da Tbox e da Abox. (ii) A segunda fase será focada na definição da ontologia. A especificação da ontologia terá como base de construção as três ontologias apresentadas na secção 3.2. As ontologias *Event*, LODE e Model F vão ser as ontologias referência para a especificação da base de conhecimento que se pretende criar neste trabalho. A terceira fase consiste na implementação de uma biblioteca aplicacional que demonstra a utilização da ontologia deste trabalho. Esta biblioteca aplicacional tem como objetivo permitir o registo e recolha de eventos aplicacionais e que seja partilhada e reutilizada por vários sistemas e que cumpra a TBox definida e será apresentada no capitulo 5.

4.2 Ecossistema da NOS Inovação

Para a recolha de informação para processamento analítico, é necessário conhecer o ecossistema de serviços e sistemas que dá suporte ao negócio de TV da NOS Inovação. Conhecer os sistemas que vão ser a origem dos *logs* semânticos é importante porque esses sistemas, suportam os casos de utilização que se pretendem descrever extraindo métricas de consumo e de monitorização.

O ecossistema existente apresentado na Figura 12 é composto por três tipos de sistemas que suportam a oferta da NOS nesse âmbito: (i) Aplicações cliente (ii) Gestão de consumo TV e OTT dos clientes (iii) Gestão do catálogo de oferta EPG e VOD.

As aplicações cliente, às quais o cliente da NOS tem acesso, são disponibilizadas nas STBs e em suporte OTT e para o seu correcto funcionamento, essas aplicações acedem aos serviços de consumo. As aplicações das STBs comunicam com os sistemas *FrontEnd NextGen* pela rede interna da NOS. As aplicações OTT caracterizam-se por estarem instaladas em dispositivos móveis do cliente com acesso à internet, sendo necessário existir um nível de segurança extra, comparando com o acesso directo que as STBs têm aos mesmos serviços. Esse nível extra de segurança é feito através de um barramento de autorização disponibilizado pela APIGEE[10]. A APIGEE tem a responsabilidade de controlar o acesso às APIs dos serviços da NOS: (i) Autorizar o acesso às APIs, (ii) Mapear endereços públicos para os URIs privados das APIs (iii) Preencher os cabeçalhos HTTP dos pedidos com contexto do utilizador, de rede do utilizador e da aplicação que o utilizador está a consumir.

Os sistemas agrupados no segmento de gestão de consumo TV e OTT da Figura 12, suportam as regras de negócio existentes nas aplicações cliente. Nesses sistemas são guardados os dados das contas de cliente, o portfólio de cliente, são processadas as regras de negócio associadas ao consumo desses clientes, como por exemplo, as gravações que cada cliente faz e as compras no vídeoclube. Neste grupo de sistemas existem ainda sistemas responsáveis por indexar todos os conteúdos existentes permitindo ao utilizador a realização de pesquisas de conteúdos. Todos estes casos de utilização são orquestrados por um sistema que é acedido pelas aplicações clientes, abstraindo das aplicações clientes a existência de todos esses serviços. Para enriquecer os sistemas de consumo de conteúdos actualizados, existem os sistemas de gestão de catálogo. Estes sistemas são responsáveis por manter atualizada a informação da grelha televisiva, do vídeoclube e dos conteúdos relacionados. Essa informação inclui, entre outros, o nome, a descrição, a duração, o género, os actores, as imagens, e outros programas associados.

Esta informação é mantida actualizada através de processos de importação de dados de diferentes fontes, orquestrados por processos sustentados em sistemas geridos por pessoas especializadas que gerem os conteúdos.

[10] https://apigee.com/api-management/

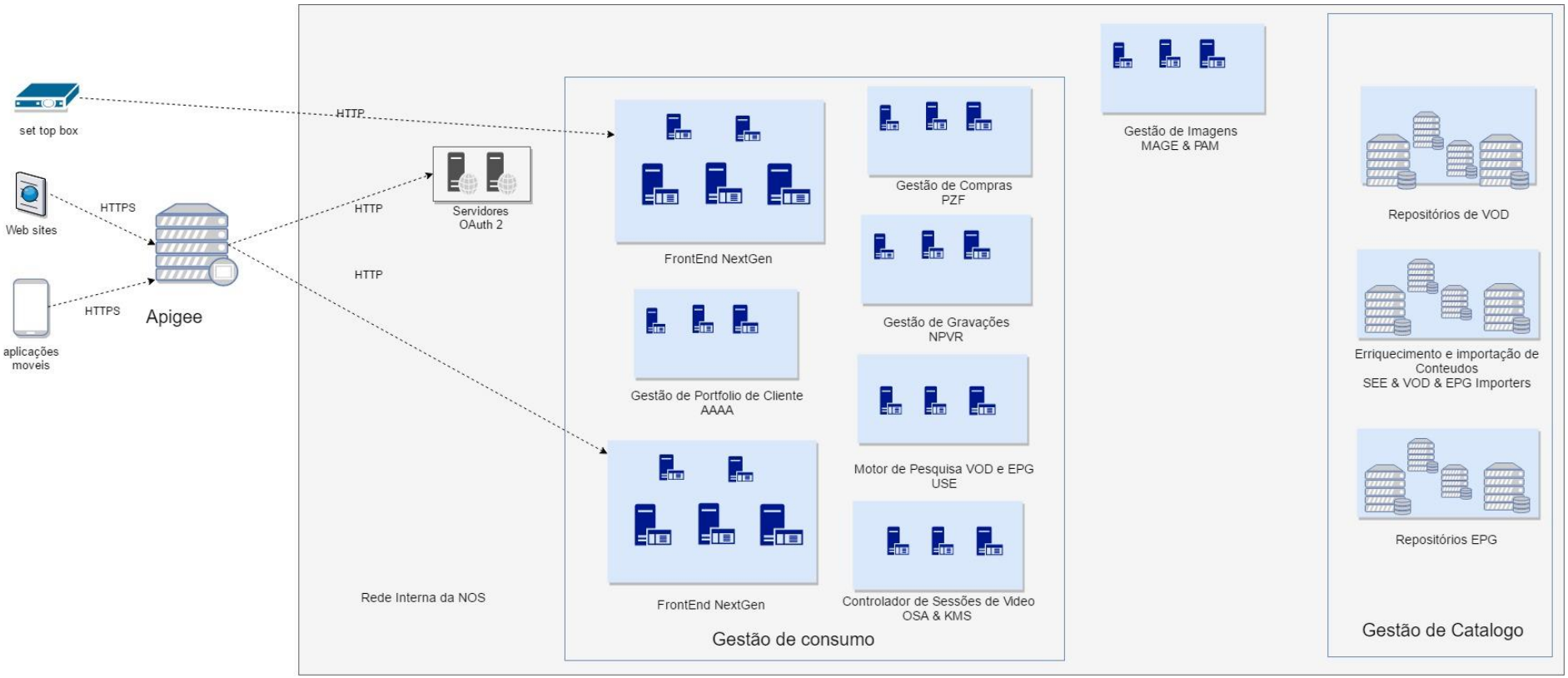

Figura 12 - Ecosistema de serviços TV e OTT

Tabela 5 - Lista de serviços TV e OTT

Âmbito	Nome	Descrição
Gestão de Consumo	FrontEnd NextGen	Serviços com APIs focadas nos casos de utilização consumidas pelas aplicações. Este serviço utiliza os restantes serviços listados.
Gestão de Consumo	AAAA – Account, Authentication, Authorization, Auditing	Serviço que gere contas de clientes, o portfólio comercial de consumo e utilizadores associados às contas de cliente
Gestão de Consumo	PZF – Purchase Zon Fondation	Serviço que gere compras de VODs
Gestão de Consumo	NPVR – Network Program Video Recording	Serviço que gere as gravações realizadas pelo cliente de conteúdos do EPG
Gestão de Consumo	USE – Unified Search Engine	Motor de pesquisa por conteúdos VOD e EPG e conteúdos relacionados
Gestão de Consumo	OSA – OTT Session Authorization & KMS –Keys Management System	Gestão de Streams de vídeo e de direitos de visualização sobre essas streams de vídeo
Gestão de Consumo e Gestão de Catálogo	PAM – Picture Asset Manager e MAGE -	Serviços de gestão de imagens. O PAM é o repositório de imagens e o MAGE responsável pela transformação de imagens.
Gestão de Catálogo	Repositórios de VOD	Conteúdo VOD inserido por uma equipa de *backoffice*
Gestão de Catálogo	Repositório de EPG	Conteúdos fornecidos pelos canais de televisão geridos em *backoffice*
Gestão de Catálogo	SEE VOD & EPG Importers	Serviços responsáveis por enriquecer os conteúdos dos repositórios.

Os serviços existentes apresentados na Figura 12 e listados na Tabela 5 são os sistemas que actualmente suportam o negócio de TV da NOS e onde ocorrem operações despoletadas pela

utilização das aplicações por parte dos clientes. Para perceber como estes sistemas interagem entre todos, segue na secção 4.3 uma lista de alguns casos de utilização. Com esta lista de casos de utilização vão ser identificados requisitos para que seja possível registar *logs* semânticos descritos com a ontologia deste trabalho para serem utilizados no processo CEP.

4.3 Casos de utilização

A ocorrência de um caso de utilização de negócio é a agregação dos casos de utilização de cada um dos sistemas envolvidos. Esta ontologia esta a ser construída com o objetivo de registar a ocorrência desses casos de utilização para permitir a descrição da ocorrência de um caso de utilização de negócio como um todo, mas ao mesmo tempo ter visibilidade do que aconteceu em cada um dos sistemas. O resultado da execução dos eventos granulares de cada um dos sistemas possibilitam a deteção e/ou validação da ocorrência de eventos de negócio através de padrões pré-definidos com sucesso ou insucesso. A definição da ontologia que é descrita por domínios de conhecimento que estão diretamente relacionados com a execução de uma acção num serviço ou sistema é uma mais-valia na validação do funcionamento dos serviços. Para demonstrar esta importância segue a análise de um conjunto de casos de utilização relevantes na organização. Esta análise apresenta a descrição dos serviços e sistemas envolvidos e da importância de cada um dos intervenientes e qual a importância que um *log* desenvolvido com esta ontologia tem na detecção de problemas, ou para validar que tudo acontece como esperado. Os casos de utilização são:

1. Navegar no EPG,
2. Agendar uma gravação
3. Subescrever um canal de televisão *premium*
4. Navegar no vídeoclube
5. Alugar um VOD no vídeo clube

4.3.1 Navegar no EPG

A navegação pela vista de canais é uma das funcionalidades que a NOS disponibiliza aos seus clientes. A lista de canais, designada por EPG, disponibiliza ao cliente um conjunto de canais aos quais o cliente consegue aceder dependendo do seu produto comercial que subescreveu. A navegação no EPG acontece a dois níveis: (i) Navegação na lista de canais. (ii) Navegar na grelha televisiva de um determinado canal.

A navegação no EPG é um caso de utilização que utiliza três serviços para disponibilizar ao cliente a lista de canais:

1. *FrontEnd NextGen* – Orquestrador do caso de utilização utilizado pelas aplicações clientes
2. Repositório EPG - Sistema que mantém informação dos conteudos do EPG actualizados

3. AAAA – Sistema que mantém informação do portfolio dos produtos comerciais do cliente.

Para que o EPG apareça atualizado na STB ou em qualquer aplicação OTT que apresente o EPG, o sistema *FrontEnd NextGen* aplica o fluxo da Figura 13:

1. Existir o pedido HTTP da aplicação cliente ao *FrontEnd* para obter os canais
2. Ter ocorrido no sistema *FrontEnd* um mecanismo de *pooling* que acede ao repositório de EPG num intervalo de tempo configurado a consultar a lista de EPG mantendo em memória a lista total de canais e a sua caracterização.
3. Por cada canal obtido validar no sistema AAAA se o canal pode ser disponibilizado ao cliente

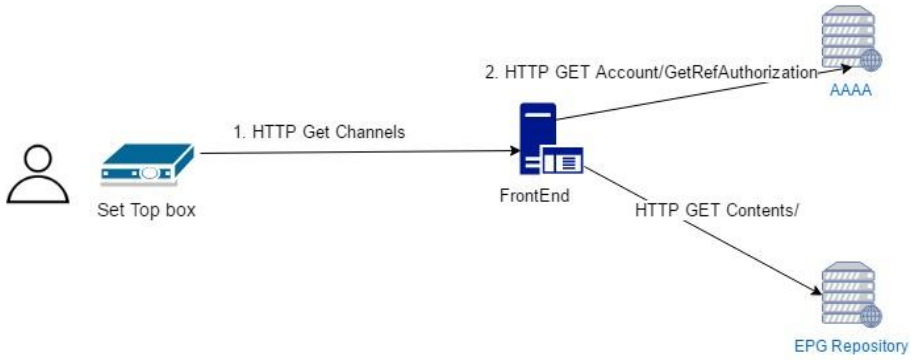

Figura 13 - Listar EPG

Associado a este caso de utilização, serão registadas ocorrências de eventos aplicacionais, um por cada pedido HTTP em cada serviço. Estes pedidos estão desfasados no tempo mas têm de acontecer para que o processo analítico consiga concluir que este caso de utilização ocorreu com sucesso. Os eventos que terão de ser gerados são:

1. Registar a ocorrência do pedido GET Channels ao *FrontEnd*
2. Registar a ocorrência do pedido GET RefAuthorization ao sistema AAAA
3. Registar a ocorrência do pedido periódico GET contents ao *EPG Repository*.
4. Registar as atualizações ao dados do sistema *EPG Repository*

Se um cliente reportar problemas associados ao EPG, tais como:

1. O detalhe de um evento de um programa não está preenchido,
2. Não aparece nenhum canal de televisão do seu portefólio

O processo analítico composto pelos domínios de conhecimento identificados anteriormente será relevante a identificar o problema com o maior nível de granularidade possível, para tal, o seguinte conjunto de passos pode ser realizado:

1. Construir um evento complexo que agregue os eventos 1, 2, 3 e 4.
    a. Para garantir que os eventos analisados sejam os deste caso de utilização, a interrogação a realizar deve garantir que os eventos obtidos são do cliente que

reportou o problema, e se estão dentro da janela temporal a que o cliente informou que a anomalia aconteceu.

2. Caso esta interrogação retorne resultados, devem-se tentar agregar os eventos, no mínimo quatro, para validar se ocorreu tudo o que é esperado.
3. Caso o resultado desta agregação não resulte no número esperado de eventos então podem existir problemas num dos serviços envolvidos.
    a. A detecção deste problema pode originar uma investigação sobre o bom funcionamento do serviço, ou da máquina que suporta esse serviço.
    b. Neste ponto pode-se fazer interrogações ao repositório de eventos registados para determinar o resultado dos eventos associados aquele serviço, ou máquina e perceber se o resultado dos eventos tem ocorrido erros ou no limite se não existem eventos registados daquele serviço ou máquina.
4. Caso o número de eventos resultado da interrogação seja igual ou maior ao esperado, deve-se garantir que existem eventos de cada um dos sistemas envolvidos
5. Deve-se validar que todos os resultados das acções são de sucesso
    a. Caso existam eventos com erro, o processo de investigação pode ter isso em conta e perceber o que originou aquele evento ter terminado com erro.
    b. Numa comunicação HTTP a falta de resposta de pedidos dentro do intervalo de tempo especificado nos *headers* da mensagem, faz com que não ocorra registo de eventos no serviço chamado. Esta é uma das causas para o facto de falhar o registo de eventos. A ocorrência de pedidos sem resposta em tempo útil, pode revelar problemas de comunicação de rede, ou que o servidor onde está o serviço esteja sobre sobrecarga e não consiga processar todos os pedidos HTTP que receba.

4.3.2 Aceder a um canal de televisão

O acesso à grelha de um canal de televisão funciona de forma diferente entre a STB e OTT como apresentado na Figura 14.

Se o cliente estiver a ver televisão por OTT os sistemas usados são:

1. *FrontEnd*
2. Epg Repository

Sempre que o utilizador quiser aceder a um canal para consultar a sua grelha de eventos ou ver tv no evento actual resulta no seguinte fluxo:

3. A aplicação cliente OTT faz um pedido HTTP ao *Frontend* para obter o detalhe da grelha de programas do canal e dos seus de eventos entre um intervalo de datas que englobe o dia actual e os sete dias anteriores.

4. No *Frontend* existe a consulta em memória dos conteúdos do canal, carregada previamente pelo processo de carregamento de conteúdos de EPG retornando a lista completa dos conteúdos desse canal para esse dia.

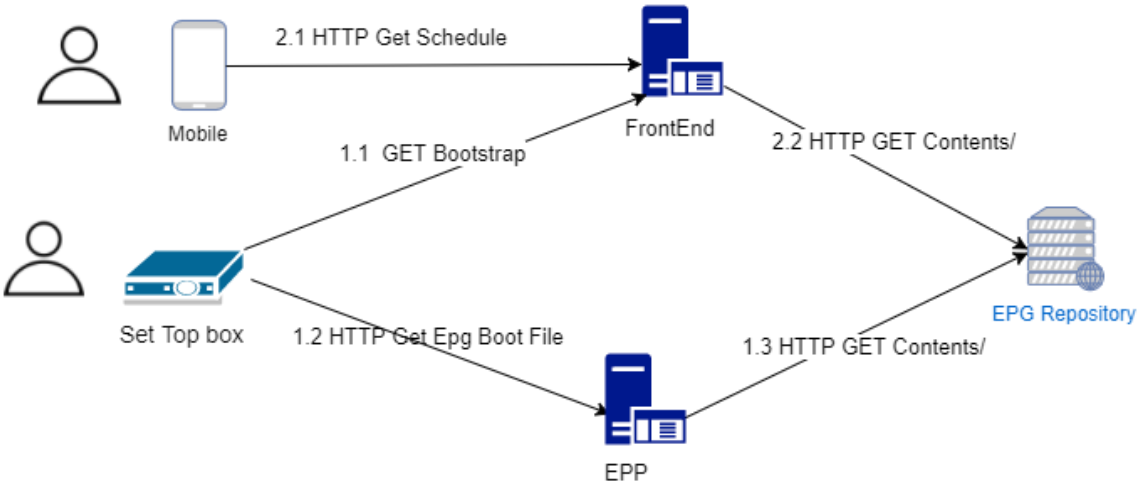

Figura 14 - Aceder a um canal

No caso da STB, o fluxo é diferente, e os sistemas envolvidos são:

1. *FrontEnd*
2. EPP – Sistema desenvolvido pelo fabricante do *software* da STB. Este sistema é responsável por obter toda a informação do cliente para que quando o cliente comece a ver televisão tenha o EPG todo disponível para ser consultado.
3. *EPG Repository*

A interação entre os sistemas ocorre da seguinte forma:

1. Quando a STB invoca o GET Bootstrap ao *FrontEnd* obtém informação para começar a funcionar, nomeadamente informação de sinal, localização onde está o ficheiro EPP, e identificação da conta do cliente.
2. Obter o ficheiro EPP que foi previamente gerado a partir do *EPG Repository*. Este ficheiro devolve todo o EPG e listagem dos eventos de cada um dos canais.

Neste cenário os eventos aplicacionais que são gerados variam dependendo da plataforma onde é realizado. Para validar se um cliente consegue ter acesso à grelha de eventos de um canal de televisão os eventos relevantes de registar são diferentes para as duas plataformas.

OTT:

1. Registo da ocorrência do *GET Schedule* no *frontend*

2. Registo da ocorrência do *GET Contents ao Epg Repository* que teve como origem o sistema de *FrontEnd*

STB:

1. Registo da ocorrência da criação do ficheiro EPP
2. Registo da ocorrência do GET *Bootstrap* ao *FrontEnd*

Com estes eventos registados, mantém-se informação relevante dos domínios de conhecimento que permitem validar se o EPG construído é válido, nomeadamente se a data e hora a que foi pedido está coerente com a última data e hora a que foi gerado no *EPG Repository*. Nos casos de utilização onde a acção dos vários participantes é assíncrona e não causal, a abordagem de correlação por informação de tempo é uma abordagem importante. Mantendo referência de quando é que os eventos aconteceram, quando é que acabaram e a sua duração, permite identificar situações como:

1. Se a data e hora a que foi pedido por um cliente aconteceu depois da actualização do *EPG Repository*, garantindo assim que o cliente está a visualizar a informação correcta.
2. Ao registar a duração do processo de importação do *EPG Repository* permite identificar se o processo está a ter uma duração anómala face às outras ocorrências deste processo.

O processo de obter informação completa do EPG é feita no âmbito do produto UMA e é consumida por dois processos: (i) *FrontEnd NextGen* (ii) Processo de criação do ficheiro EPP. Para validar se estes dois processos estão a consumir a informação, é importante associar ao registo dessa ocorrência informação de qual foi o sistema que pediu essa informação. Um processo analítico acrescenta valor a este processo de negócio se monitorizar se os sistemas que devem pedir o GET Contents, o estão a fazer e se o estão a fazer num intervalo de tempo esperado, comparando o intervalo de data e hora a que acontece e se o processo de importação das várias fontes do *EPG Repository* ocorrem como esperado.

4.3.3 Agendamento de uma gravação

O agendamento de uma gravação de um programa que ocorra num canal de televisão, é um use case que possibilita ao cliente agendar a gravação de um único programa de televisão, ou caso seja possível da serie completa desse programa.

Este caso de utilização necessita de 3 serviços:

1. FrontEnd NextGen
2. NPVR
3. AAAA

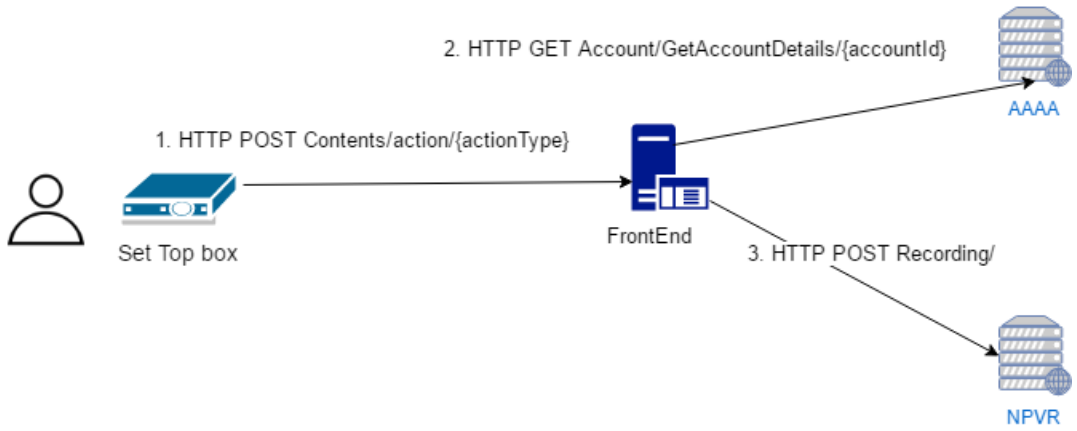

Figura 15 - Realizar uma gravação

A realização de uma gravação requere:

1. O cliente tenha a acção de gravar disponível no programa de televisão onde esta prestes a realizar a acção
2. O utilizador possua na sua conta de subscrição de cliente uma STB.

O Fluxo de agendamento da gravação consiste em:

1. Invocação da aplicação cliente ao sistema *Frontend*
2. O sistema *FrontEnd* valida se a conta cliente tem uma STB na sua conta de cliente
3. O pedido de agendamento ao sistema NPVR aconteça com sucesso

Associado a este caso de utilização, serão registadas ocorrências de eventos aplicacionais, um por cada pedido HTTP que têm de acontecer para que o processo analítico consiga concluir que este caso de utilização ocorreu com sucesso. Os eventos que terão de ser gerados são:

1. Registar a ocorrência do pedido POST Contents/action/{actionType} ao *FrontEnd*
2. Registar a ocorrência do pedido GET Account/AccountDetails ao sistema AAAA
3. Caso o ponto dois aconteça com sucesso, é registada a ocorrência do pedido POST Recording.

A ocorrência do pedido identificado pelo ponto 1 despoleta as restantes duas chamadas, ao contrário dos casos de utilização anteriores este caso de utilização é composto por ocorrências síncronas, ou seja, o primeiro passo só acaba quando os que foram iniciados por ele também acabarem. Comparando com os anteriores, saber se o cliente consulta EPG actualizado e válido, consiste em correlacionar ocorrências de eventos não causais e desacoplados no espaço e no tempo. Neste cenário saber que uma gravação ocorreu com sucesso é uma sequência de acções causais, o passo 1 causa a ocorrência do passo 2 e só se este retornar os dados pretendidos é que acontece o passo 3. Para saber se este caso de utilização aconteceu com sucesso, basta analisar o resultado da ocorrência do ponto 1, só se este retornar erro, é que pode ser relevante avaliar os restantes dois e perceber onde ocorreu o problema.

4.3.4 Subscrição de um canal premium

A subscrição de um canal *premium* é um caso de utilização que utiliza sistemas da NOS Inovação e do departamento que gere toda a infraestrutura tecnológica de relação com o cliente e faturação, e portanto parte do caso de utilização de negócio sai fora do âmbito dos sistemas apresentados na secção 4.1. Interagir com um sistema que não controlamos o seu desenvolvimento onde não temos a capacidade gerar eventos que descrevem que eles ocorreram, é necessário correlacionar os seus pedidos através dos dados que estão a ser usados. Quando um cliente navega no EPG na sua STB, os canais que são *premium*, encontram-se bloqueados por omissão. Caso o cliente queira desbloquear um desses canais deve faze-lo por um dos canais que a NOS disponibiliza: (i) Call Center (ii) Site da NOS na zona de cliente (iii) Na STB na posição do canal utilizando o comando. O processo que será descrito consiste no processo que utiliza o canal (iii).

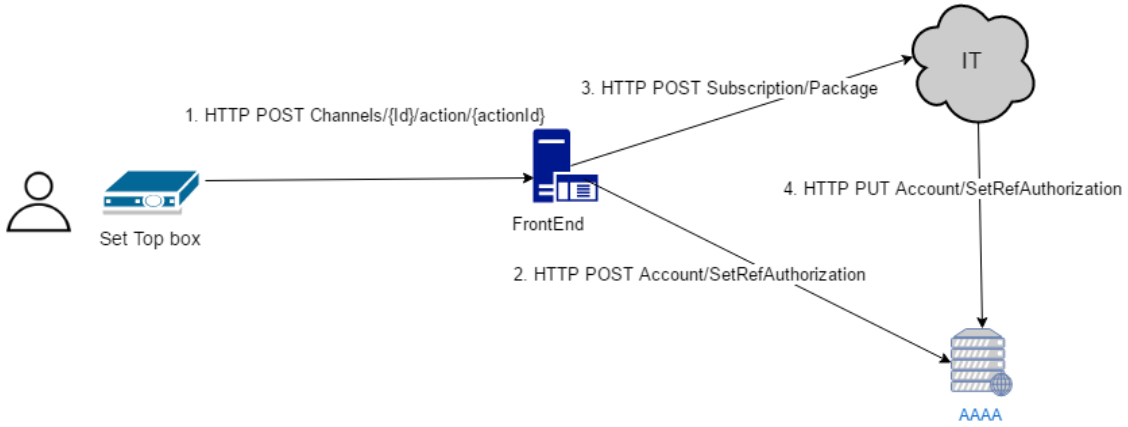

Figura 16 - Subscrição de canal premium

Este caso de utilização de negócio utiliza dois sistemas da NOS Inovação:

1. FrontEnd
2. AAAA

O Fluxo da subscrição do canal consiste nos seguintes passos:

1. O cliente executa com o comando da televisão a acção de subscrever canal
2. No passo 1, A STB envio para o *FrontEnd* o pedido com essa indicação, parametrizando no URL o Id do canal e o Id da acção, neste caso – subscrever canal.
3. No passo 2 por forma a dar serviço ao cliente o mais cedo possível desse canal, é dada uma autorização temporária ao cliente para que consiga ver o canal de televisão de imediato
4. No Passo 3, É submetido o pedido aos sistemas do IT, que assincronamente vão proceder ao registo da subscrição
5. E por último caso a subscrição seja possível de ser realizada, é executado o passo 4. O sistema do IT responsável por comunicar com os sistemas da NOS Inovação faz uma invocação ao sistema AAAA colocando como permanente a autorização da visualização

do canal, Caso a autorização não seja confirmada pelo IT, a autorização temporário expira e o cliente deixa de ter acesso ao canal.

A operação de cancelamento da subscrição consiste na invocação do passo 4 mas com o método HTTP de DELETE.

Os eventos que serão gerados, tal como nos anteriores, corresponde a cada uma das interações que existe entre os serviços, no entanto um dos registos não é vinculativo de que o caso de utilização tenha terminado com sucesso. O ponto 2 apesar de dar direitos para o cliente aceder temporariamente ao canal, esta autorização só é válida até que o IT valide se o cliente tem condições financeiras, Se não tiver o IT invoca o HTTP DELETE e a autorização é removida do cliente, deixando de haver registo que esse produto teve alguma vez no cliente. Uma vez que o ponto 2 realiza algo que tem uma validade temporária, poderia pensar-se que seria desnecessário registar o ponto 2 mantendo só o registo da resposta do IT, mas isso não é verdade. É importante manter o registo de todos os eventos para garantir que a autorização temporária foi dada, e se nesse intervalo de tempo que a autorização for válida o IT não responder com a resposta final o cliente deixa de ter acesso à subscrição. Desta forma o processamento analítico com essa informação consegue identificar que o cliente não consegue visualizar o canal porque o sistema do IT não respondeu a tempo, em vez o cliente está perante alguma situação de infração financeira com a NOS. Este caso de utilização demonstra a importância que existe em registar todas as ocorrências para garantir que cada um dos sistemas aconteceu.

4.3.5 Alugar e comprar VOD no vídeoclube

O Aluguer e compra de um VOD, à semelhança com a subscrição de um canal premium interage com o IT para a submissão da acção. O Aluguer consiste em associar ao portfolio do cliente o aluguer de um VOD que fica disponível para consumo durante 48h. No caso da compra o VOD fica associado ao portfolio do cliente sem data de expiração. Este use case requere que primeiro tenha sido executado com sucesso a navegação até ao conteúdo onde esta a ocorrer a acção, existindo assim uma relação entre os dois casos de utilização.

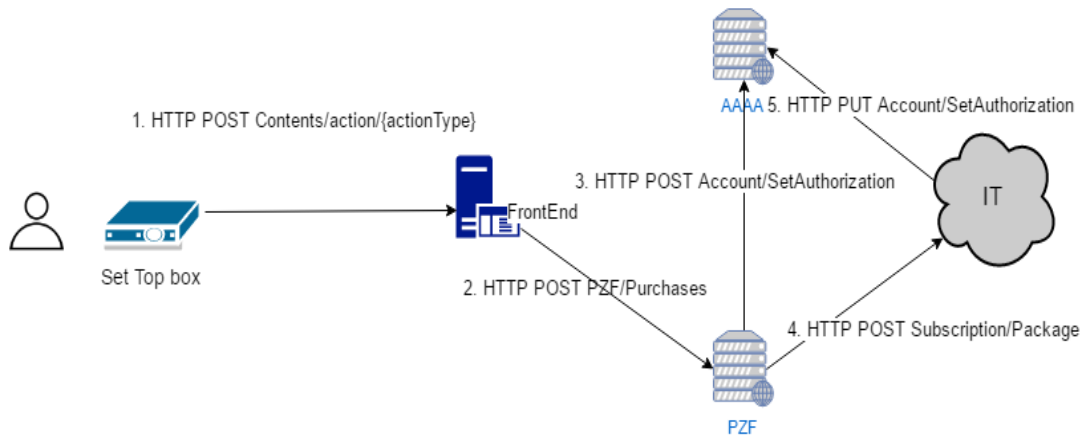

Figura 17 – Aluger ou compra de VOD

O Fluxo destas duas operações consiste nos seguintes passos:

1. O cliente executa com o comando da televisão a acção de aluguer/compra de VOD
2. No passo 1, A STB envio para o FrontEnd o pedido com essa indicação, parametrizando no URL o Id do canal (contentId) e o Id da acção, neste caso – alugar/comprar VOD.
3. No passo 2, é enviado um pedido ao PZF para a realização da acção da compra / aluguer
4. No passo 3, é feita a associado do VOD com uma data de expiração no portfolio do cliente
5. No passo 4, é enviado o pedido de confirmação ao IT para validar a compra ou o aluguer
6. No passo 5, é feita a confirmação da compra, caso não seja valido, a data temporária da compra expira e o VOD é retirado da conta do cliente.

Semelhante ao caso de utilização 4.3.4, a compra ou aluguer no vídeoclube também depende dos sistemas de IT

A análise dos casos de utilização teve como principal objetivo expor a complexidade que existe na execução dos casos de utilização e nos vários tipos de relações que existem entre as ocorrências em cada um dos sistemas. Esta análise serviu para apresentar a importância que existe em registar dados numa ocorrência associados com diferentes domínios de conhecimento, tais como referências temporais, sistemas envolvidos, redes de comunicação usadas pelos clientes, e resultado das operações nos sistemas.

Outra constatação que se retira desta análise é o protocolo de comunicação maioritariamente usado entre os sistemas. A interação entre os vários sistemas, é feita maioritariamente sobre HTTP. Este protocolo de comunicação consiste num modelo pedido, resposta síncrono que não persiste estado, onde as acções a executar são identificadas por um método HTTP[11], um URL único e permite o envio de dados do pedido em cabeçalhos separados do próprio corpo da mensagem em que o resultado das operações tem sempre um código de resultado associado[12]. Todas estas características faz deste protocolo um meio de comunicação também ele semântico mas não semântico o suficiente para representar ocorrências de acções nos serviços e sistemas. Para enriquecer o processamento CEP, na descrição semântica do acontecimento de eventos aplicacionais, bem como a descrição do domínio de informação associado e no resultado de uma operação, como resultado da análise dos principais casos de utilização, verifica-se que a ontologia deve ser composta por domínios de conhecimento que representem os seguintes conceitos:

1. Um evento aplicacional simples é um acontecimento que ocorre numa instância de um serviço/sistema que está envolvido numa cadeia de invocações entre serviços/sistemas sequenciais ou paralelas.
2. O registo de um caso de utilização completo consiste no registo de todos os eventos simples. Será da responsabilidade do sistema de processamento analítico a construção de eventos complexos, através de operações de composição, agregação ou interrogação.

[11] https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
[12] https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

3. As acções que são realizadas têm sempre um sujeito associado, ou seja, uma entidade sobre a qual se está a realizar uma operação e ainda um conjunto de parâmetros que condicionam a resposta das acções. Para enriquecer o processo CEP, a representação dessa entidade deverá ser homogénea baseada em conceitos e termos definidos na ontologia dos quais se consiga representar semanticamente o sujeito, ou os sujeitos sobre os quais a acção aconteceu.
4. A descrição temporal de uma ocorrência de uma acção num serviço ou sistema é também uma necessidade. Verificou-se que a capacidade de conseguir correlacionar a ocorrência de acções pela sua data de criação ou finalização ou pela sua duração permite, não só correlacionar eventos em janelas temporais, mas também identificar acções cuja sua execução esteja a demorar mais do que o previsto.
5. A execução de uma acção, gera em várias situações um resultado, resultado esse que representa objectos do domínio aplicacional. Através de uma representação homogénea baseada em conceitos e termos definidos na ontologia, é relevante representar o resultado de uma operação através da descrição do modelo de domínio envolvido.

A Figura 18 representa uma primeira versão do mapa de domínios de conhecimento que deverão existir na nossa ontologia identificados da análise feita aos casos de utilização. Um domínio de conhecimento é representado por um retângulo com traço alternado que representa que o domínio ainda não está fechado na sua definição. Os conceitos de cada um dos domínios são representados por círculos ovais, baseando essa grafismo no desenho de tripletos de RDF. Para os conceitos que já conhecida uma relação é especificada através de um traço.

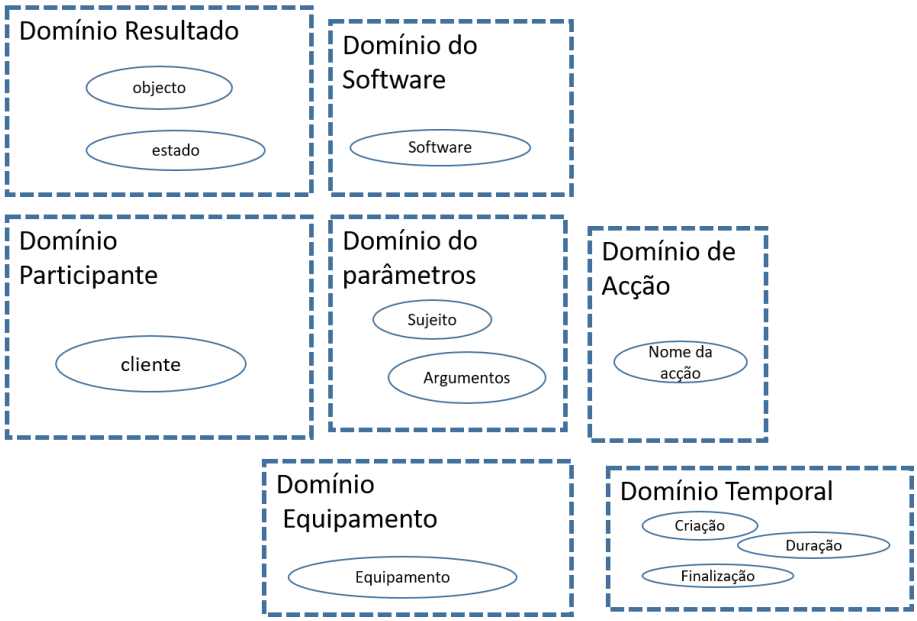

Figura 18 - Domínios de conhecimento identificados dos casos de utilização

Nas próximas secções este mapa de domínios de conhecimento vai sendo atualizado à medida que surjam novos conceitos associados a cada um dos domínios.

4.4 *Log* actual

Atualmente, a recolha de dados que registam o acontecimento de acções nos serviços e sistemas, consiste na escrita em ficheiros de texto com uma estrutura semiestruturada apresentada no capítulo 2. O conteúdo que é escrito nesses ficheiros contém informação que é relevante para o processamento analítico, mas a falta de estrutura e semântica do seu conteúdo tornou o processo analítico complexo e extenso. Para resolver este problema e outros identificados no capítulo 2, este trabalho focou-se em rever a utilização desse log e evoluir para um registo estruturado e com expressividade semântica. Para não quebrar a compatibilidade dos dados que são usados no processamento analítico e tirar o máximo partido daquilo que já existe disponível à data de hoje nos sistemas a monitorizar para a escrita deste *log*, é necessário identificar todos os campos que são registados atualmente, e qual o seu significado. Para esse estudo, considere-se o exemplo de escrita de um *log* actual apresentado na Listagem 7.

Este *log* é composto por vários campos, escritos por componentes diferentes e por isso, os caracteres separadores do domínio de escrita de cada um desses componentes tomam importância da forma como são usados. Cada uma das secções deste *log* é escrita por componentes desacoplados durante a execução da rotina de código e para definir a fronteira de cada um dos componentes que participam na escrita deste *log*, cada um escreve os seus valores, entre dois caracteres especiais, [e]. Estes dois caracteres formam um espaço próprio onde são registados os dados que o componente souber registar. Sendo que os componentes que registam dados para os log são independentes e desacoplados uns dos outros, faz com que ocorram registos repetidos e que o formato global do log não seja homogéneo. Como por exemplo o nome do servidor aparece registado duas vezes e o mac do equipamento do cliente aparece também registado em dois locais no *log*.

A primeira secção do *log*, escreve os valores separados por | sem que sejam identificados, existindo uma convenção sobre o seu significado consoante a posição onde se encontram, descrito na Tabela 6.

Tabela 6 - Primeira secção do log actual

Posição	Descrição	Valor da Listagem 7
Primeira	Data e hora local onde o log foi registado	[2017-04-24 14:00:59,969
Segunda	Nome do componente que registou os valores	NOS.Logging.log4net.PostSharp.FEFormat.EntryPointLoggingAttribute
Terceira	Severidade do Log[13].	INFO
Quarta	Identificação da thread que estava em execução	99

[13] A severidade do log corresponde aos níveis disponíveis na infra-estrutura Log4Net

```
[2017-04-24 14:00:59,969|
NOS.Logging.log4net.PostSharp.FEFormat.EntryPointLoggingAttribute|
INFO|
99]
[{nodeItemId=(null),
expand=(null),
skip=(null),
 log4net:UserName=IIS APPPOOL\NextGenAppPool,
 log4net:Identity=, expandVersion=(null), reqId=(null), top=(null),
 log4net:HostName=SVLNDIDFE27, expandDepth=(null)
}]
:00d0375d1031;
SVLNDIDFE27;
StoreController.StoreChildren;
GET /NextGen.FE/api/store/vodcategory.editorial@uma.185779065/children/items;
1.0.0-RC1b;4;
[
X-Core-UserType:profile|
X-Core-UserId:?|
X-Core-DeviceId:00d0375d1031|
X-Core-DeviceType:stb|
X-Core-AccountId:s801004230|
X-Core-Language:por|
X-Core-AppId:NEXTGEN_E|
X-Core-AppVersion:1.0.0-RC1b|
X-Core-AppFeatures:?|
X-Core-UserProfileId:501113|
X-Core-Profile:?|
X-Core-UserDisplayName:?|
X-Core-UserSessionId:?|
X-Core-NetworkId:42833|
X-Core-VideoSessionId:?|
X-Core-ClientIp:10.147.26.34|
X-Core-Username:?
];false||
```

Listagem 7 – Exemplo *log* actual

A segunda secção, num formato proprietário, apresenta os valores dos argumentos da rotina de código que está a ser invocada, associada a um nome identificativo, formando assim pares chave/valor separados pelo simbolo = no registo dos campos.
A terceira secção, detalhada na Tabela 7 contém dados que caracterizam a identificação da rotina de código que esta em execução.

Tabela 7- Terceira secção do log actual

Posição	Descrição	Valor da Listagem 7
Primeira	MAC do equipamento que o cliente está a usar	00d0375d1031
Segunda	Nome do servidor onde ocorreu o registo	SVLNDIDFE27
Terceira	Nome do componente aplicacional onde ocorreu o registo	StoreController.StoreChildren
Quarta	Informação HTTP do pedido que foi feito que originou o registo	GET NextGen.FE/api/store/vodcategory.editorial@uma.185779065/children/items;
Quinta	Versão da aplicação onde o log foi registado	1.0.0-RC1b

Na quarta secção são registados os cabeçalhos HTTP aplicacionais. O registo destes dados apresentados na Tabela 9, Tabela 10, Tabela 11, Tabela 12, Tabela 13 segue um formato pares chave/valor, onde os valores são caracterizados pela sua chave.

Tabela 8 - Dados do cliente que executou o caso de utilização

Chave	Descrição
X-Core-UserType	Tipo de Identificação mais concreta disponível nos vários campos de identificação de cliente
X-Core-UserId	Identificador único do utilizador OTT
X-Core-AccountId	Identificação da conta de cliente do utilizador
X-Core-Username	Identificado do nome de utilizador
X-Core-UserProfileId	Identificação do perfil do utilizador
X-Core-Profile	Nome do perfil
X-Core-UserDisplayName	Nome do utilizador

Tabela 9 - Dados do equipamento usado pelo cliente

Chave	Descrição
X-Core-DeviceId	Identificação do device de onde o pedido HTTP ao serviço foi realizado
X-Core-DeviceType	Tipo de device do X-Core-DeviceId

Tabela 10 - Dados da caracterização da aplicação que fez o pedido

Chave	Descrição
X-Core-AppId	Identificador da aplicação
X-Core-AppVersion	Identificação da versão da aplicação
X-Core-AppFeatures	Listagem de funcionalidades que a aplicação suporta

Tabela 11 - Dados de rede da comunicação executada

Chave	Descrição
X-Core-ClientIp	Identificação do IP que o cliente tem a si atribuído
X-Core-NetworkId	Identificação da rede onde o cliente está ligado. Este campo só vem preenchido quando o pedido vem de uma STB

Tabela 12 - Dados de sessão existentes referentes ao cliente

Chave	Descrição
X-Core-UserSessionId	Identificador único da sessão do utilizador
X-Core-VideoSessionId	Identificação da sessão de vídeo que o utilizador está a ver

Tabela 13 - Dados de cultura configurada na aplicação cliente

Chave	Descrição
X-Core-Language	identificador da língua configurada na aplicação cliente

Como se pode verificar, a escrita do log por diferentes componentes produz um formato que têm vários problemas:

1. Heterogeneidade: A mesma informação é escrita de diferentes formas, com recurso a caracteres especiais;

2. Ambiguidade: os dados registados são ambíguos uma vez que a mesma informação é registada mais do que uma vez;

3. Omissão: Existem dados que não são registados nomeadamente o nome da aplicação onde o registo foi feito, a duração da execução, o tipo de dados usados na invocação e de que domínio aplicacional eram os dados que foram produzidos durante a execução desta rotina;

4. Valor: o registo de informação desnecessária para o processamento analítico, nomeadamente o nome dos componentes que registaram os *logs*.

Apesar do formato apresentar os problemas já identificados, os valores são relevantes. É necessário que a estrutura cumpra um formato que seja comum a todos os campos, definido por um *standard*, para permitir que a sua escrita e leitura seja mais simples sem a necessidade de interpretadores específicos para cada secção do log, simplificando e reduzindo o tempo de processamento.

Uma análise detalhada aos dados descritos anteriormente, verifica-se que os dados recolhidos podem-se agrupar em sub-domínios específicos do nosso domínio de *logging*. Este *log* contem conceitos que representam informação temporal, a identidade do participante, dados que caracterizam a aplicação que causou o *log*, e dados que caracterizam a aplicação onde este *log* aconteceu. Existem ainda dados que descrevem o resultado da operação. Desta análise podemos constatar que a definição do evento para este trabalho será baseado no log actual composto pelos domínios apresentados na Figura 19.

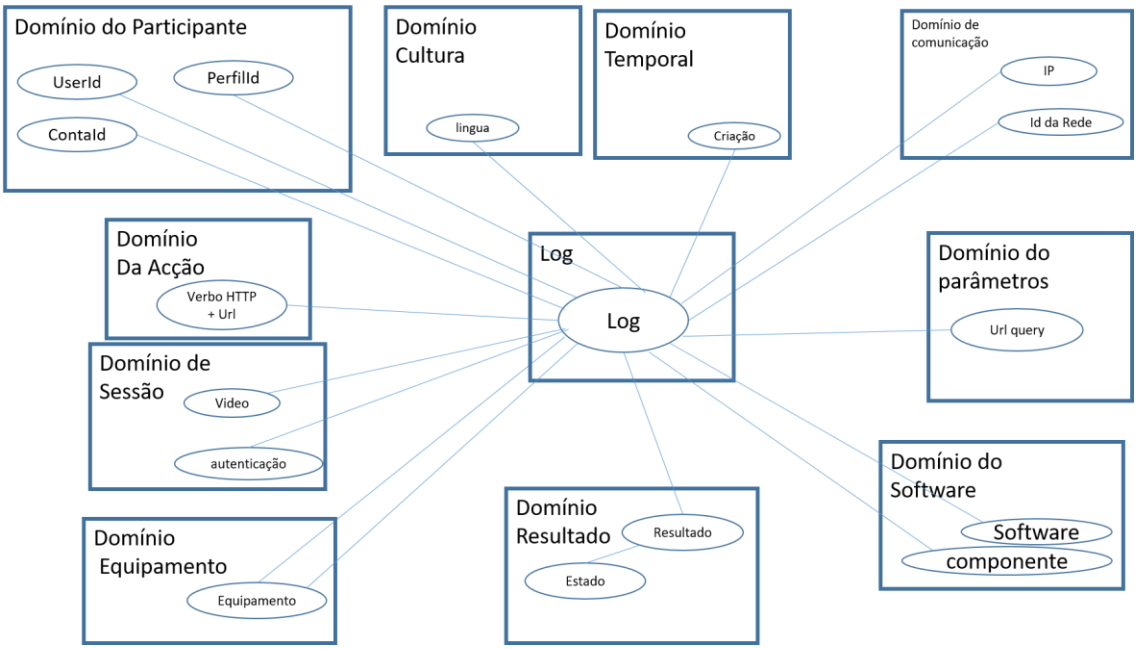

Figura 19 - Domínios do *log*

Os domínios presentes no *log* actual são:

1. Participante – Componentes que identificam o cliente
2. Temporal – Componentes temporais como a data de criação
3. Rede – Dados de comunicação usados, tais como IP do cliente, e a rede usada
4. Acção – Descrita por informação do protocolo HTTP
5. Parâmetros – Identificação dos parâmetros usados
6. Equipamento – Qual o equipamento usado, tanto da parte da aplicação que causou este evento, como a identificação do equipamento onde o serviço ou sistema estava em execução
7. Resultado – O resultado é apenas descrito com um nível de severidade
8. Software – De que *software* é que saiu o pedido HTTP que causou este log, e em que *software* é que este *log* foi escrito bem como o detalhe do nome do componente onde o *log* foi escrito.
9. Cultura – A cultura da aplicação cliente existe no log definida pela língua com que a UI apresenta o texto.

Dos domínios já identificados através da análise dos casos de utilização verifica-se que existem domínios de conhecimento já identificados anteriormente, com os que agora foram caracterizados. Em alguns casos conseguiu-se enriquecer domínios com mais conceitos, como é o caso do participante que se constata já existirem campos que caracterizam os diferentes níveis de identidade de um consumidor de conteúdos da NOS – UserId, PerfilId, ContaId. No caso do domínio de acções, o *log* actual baseia-se num conceito cujo valor é diferente do identificado nos casos de utilização. Pretende-se que o domínio de conhecimento da ontologia seja desacoplado do domínio aplicacional e portanto a definição do conceito da acção identificado nos casos de utilização será para manter. No caso do domínio de *software* o *log* actual regista o nome do componente do serviço onde o log foi registado. Analisando o *log* actual, o registo do Url que foi invocado permite identificar a rotina de código onde o registo do *log* aconteceu, mas existem cenários onde para cada rotina de um componente existe mais do que um Url configurado. Para o processo analítico ter este tipo de ambiguidade é um problema porque vai introduzir níveis extras de processamento para manter a associação dos diferentes Url à rotina de código respectiva. Face a este problema, mantém-se o domínio da acção caracterizado pelo conceito do nome da acção e o dominio de *software* ganha um novo conceito que é a rotina de código executada.

O estudo do *log* actual veio actualizar o mapa de domínios de conhecimento e respectivos conceitos como apresentado na Figura 20.

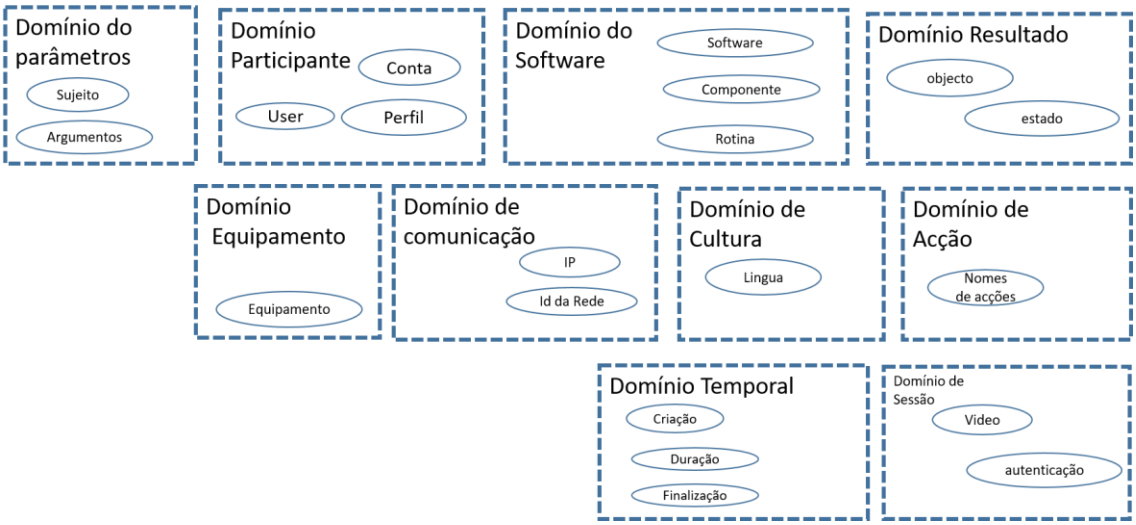

Figura 20 - Domínio de conhecimento com conceitos do *log* actual

4.5 Processo analítico actual

O processamento analítico que é enriquecido por *logs* semelhantes ao da Listagem 7, é composto por um conjunto de passos. Num dos passos de ETL do processamento analítico referido na secção 2.1, os dados depois de extraídos do ficheiro é normalizada para uma tabela com o *schema* da Listagem 8.

```
CREATE TABLE [dbo].[TMP_UMA_Logs](
        [LogDate] [nvarchar](250) NULL,
        [LogHour] [nvarchar](250) NULL,
        [s-ip] [nvarchar](250) NULL,
        [cs-method] [nvarchar](250) NULL,
        [cs-uri-stem] [nvarchar](512) NULL,
        [cs-uri-query] [nvarchar](250) NULL,
        [s-port] [nvarchar](250) NULL,
        [cs-username] [nvarchar](250) NULL,
        [c-ip] [nvarchar](250) NULL,
        [cs-useragent] [nvarchar](512) NULL,
        [sc-status] [nvarchar](250) NULL,
        [sc-substatus] [nvarchar](250) NULL,
        [sc-win32-status] [nvarchar](250) NULL,
        [sc-bytes] [nvarchar](250) NULL,
        [cs-bytes] [nvarchar](250) NULL,
        [time-taken] [nvarchar](250) NULL
) ON [PRIMARY]  WITH ( DATA_COMPRESSION = PAGE )
```

.

Listagem 8 - Schema dos *logs* no ETL

Nesta listagem podemos encontrar os dados dos domínios da Figura 19, e ainda informação que é cruzada com *logs* do IIS:

1. Sc-status - HTTP *status code*[14].
2. Cs-useragent – *software* que fez o pedido HTTP em nome de um utilizador real.
3. s-port – Porto da aplicação que estava a escuta
4. time-taken – Tempo que demorou entre o pedido e resposta
5. sc-bytes – Dimensão da resposta do pedido HTTP

Com o objectivo de perceber as acções que um utilizador fez, existe uma segunda tabela factual que é alimentada através dos dados da tabela descrita na Listagem 8. Essa tabela, da Listagem 9, contem as acções que um cliente fez ao longo do tempo e são apresentado alguns exemplos na Tabela 14.

```
CREATE TABLE [dbo].[FACT_UMA_ClientEvents](
        [MONTH_ID] [int] NOT NULL,
        [DAY_ID] [int] NOT NULL,
        [LogDate] [datetime] NOT NULL,
        [IP] [nvarchar](15) NOT NULL,
        [MAC] [nvarchar](12) NOT NULL,
        [EventType] [nvarchar](100) NOT NULL,
        [S_Port] [int] NULL,
```

Listagem 9 - Tabela factual de acções do cliente actual

Tabela 14 - Acções realizadas de um cliente

LogDate	IP	MAC	EventType
2017-04-23 00:00:00.000	10.216.155.116	003676847369	ZAPPING
2017-04-23 00:15:25.000	10.216.155.116	003676847369	EPG - PERIODIC
2017-04-23 03:15:38.000	10.216.155.116	003676847369	BOOT
2017-04-23 03:15:53.000	10.216.155.116	003676847369	ZAPPING
2017-04-23 21:56:54.000	10.216.155.116	003676847369	TW/nPVR/VOD

Do que se consegue perceber a informação das acções que acontecem são descritas com pouca informação, sendo apenas identificada a hora a que acontece, o equipamento do cliente, neste caso uma *set-top box*, o endereço IP e seu respectivo MAC e por fim o nome do evento. Este nome do evento usado não é coerente com o que foi definido pela equipa do produto e implementado no serviço que originou o *log*. Este nome do evento é obtido através de um mapeamento directo dos URLs dos serviços enviados no *log*.

[14] https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Face ao que foi apresentado nesta secção, pretende-se simplificar, normalizar e extender a informação representada pelas Listagem 8 e Listagem 9.

A simplificação consiste em definir na ontologia um conjunto de domínios coesos e desacoplados cujos valores representem a informação que o domínio diz ser, sem ser necessário processamentos extra sobre os valores para obter múltiplos valores. Normalizar consiste em definir na ontologia as instâncias de:

1. Acções documentados até hoje.
2. Termos que identificam os domínios aplicacionais existentes.

A normalização desses valores disponível na ontologia tem como objectivo a sua utilização no processo analítico em tabelas semelhantes à Tabela 14, cujos nomes dos eventos sejam uniformizados à organização, usando para isso os valores que foram definidos na documentação do produto e entregue à equipa de desenvolvimento. Por fim pretendemos enriquecer a informação de acontecimentos realizados pelo cliente, com mais dados, nomeadamente identificar as entidades de negócio envolvidas e qual o tipo delas.

Ainda no objectivo de estender o processamento analítico com mais dados, pretende-se enriquecer a base de dados com tabelas de dimensões que caracterizem os restantes domínios apresentados na Figura 19, evitando que não seja necessário recorrer a *logs* de infraestrutura como é o caso dos *logs* de IIS, se possível reduzir o número de importações de base de dados dos sistemas para processamento analítico para perceber a actividade do cliente. Para atingir este objectivo a ontologia deste trabalho terá de ser composta pelos subdomínios que representem os domínios existentes mais o que os *logs* do IIS forneciam, nomeadamente identificação do *software* do cliente (cs-useragent), código da resposta de sucesso ou insucesso (sc-status), a duração do pedido (time-taken) e ainda a dimensão em bytes da resposta (sc-bytes). Todos estes novos dados terão de ser organizados em domínios coesos, desacoplados mas que estejam todos relacionados com o evento.

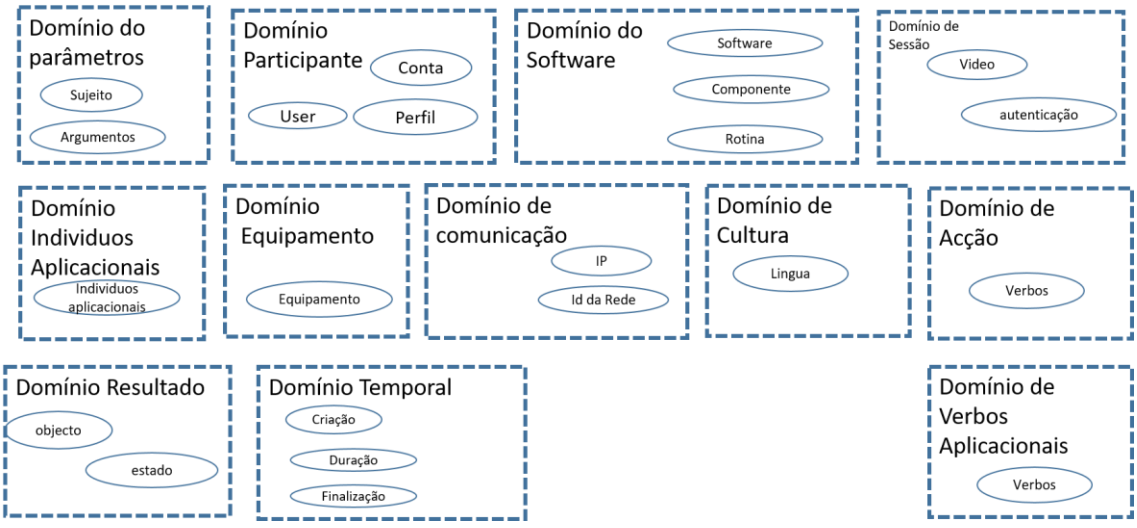

Figura 21 - Domínios do evento aplicacional

A ontologia irá suportar os domínios de conhecimento apresentados na Figura 21, que são:

1. Parâmetros – Parâmetros usados na execução da acção.
2. Participante – Caracterização dos diferentes tipos de participantes.
3. *Software* – Descrição do que é um *software.*
4. Sessão – Informação de sessão que o participante tem activa.
5. Indivíduos Aplicacionais – Conceitos do domínio aplicacional encontrado em todos os ecossistemas.
6. Comunicação – Caracterização de informação da comunicação que ocorreu.
7. Cultura – Informação da lingua usada na aplicação cliente
8. Acção – Verbo que caracteriza a acção realizada pelo participante.
9. Resultado – Caracterização do resultado da operação que originou o evento.
10. Temporal – Informação temporal do evento.

Com os domínios de conhecimento do evento identificados, a próxima secção irá apresentar o significado de cada um deles.

4.6 Domínio de conhecimento

Os sistemas do ecossistema da NOS vão ser relevantes para caracterizar domínios de informação apresentados na secção 4.2.

4.6.1 Domínio temporal

A ocorrência de um acontecimento tem associado a si referências temporais. As ontologias apresentadas neste trabalho são caracterizadas por conceitos temporais, o *log* actual contém um atributo que representa o valor da data de registo do *log* e também foi identificado pela análise dos casos de utilização. Os conceitos identificados como relevantes para a ontologia deste trabalho consiste na representação da Data e hora com o qual se consiga caracterizar o inicio e o fim de um acontecimento e do conceito de intervalo de tempo descrito como duração. O domínio temporal define portanto dois conceitos que são usados na descrição dos eventos:

1. Instante Data e hora
2. Intervalo de tempo

4.6.2 Domínio de participantes

O domínio de participantes é caracterizado pela identidade com que um utilizador possa ser identificado. Quando é criado um cliente TV da NOS é criado um identificador associado à instalação da sua casa por um identificador designado por *household*. Este identificador tem associado a si, os equipamentos que o cliente tem instalado em casa, *a set-top box* e o *router* identificados pelos seus MACs respectivos. Uma *household* também tem como finalidade relacionar todo o consumo à instalação dos equipamentos daquele casa, como por exemplo é

associada a uma *household* que se mantem todo o portfolio de produtos que o cliente subescreve, aluga ou compra.

Associado a uma instalação numa casa, *household*, existem ainda um nível de identidade que permite identificar perfis de consumo associado à *set-top box* e a aplicações OTT.

Por fim existe ainda um nível de identidade que identifica uma pessoa- *User*. Esta identidade é persistida na base de dados com informação de autenticação associado a informação de entrada nas aplicações OTT. Um *user* pode estar associado a uma *household* e consumir conteúdos do portfolio dessa *household*. No entanto existe algumas restrições associadas. Um utilizador tem níveis de permissões associado relativo ao consumo, pode ser administrador ou convidado. Caso seja administrador isso identifica que essa pessoa é o proprietário do contrato da *household*. Caso seja convidado, não pode ser administrador de nenhuma *household* e só estar associado como convidado a uma *household.* No *log* actual o participante é caracterizado pela presença de um ou mais destes valores que estão presentes nos cabeçalhos HTTP e são caracterizados por dois atributos: (i) Identificador único (ii) Tipo é inferido pelo campo que contém o valor.

1. User: X-core-UserId
2. Household: X-Core-AccountId
3. Profile: X-Core-ProfileId

Tabela 15 - Tipos de participante

Tipo	Descrição
User	Equipamento móvel
Household	Dispositivo externo que se liga a um computador ou televisão
Profile	Computador de espessura fina sem teclado

4.6.3 Domínio de parâmetros

O domínio dos parâmetros consiste na caracterização dos argumentos usados na acção que despoleta a ocorrência do evento aplicacional. Este domínio de conhecimento é definido por dois conceitos: (i) Sujeito (ii) Argumentos

O sujeito do evento serve para representar a entidade de negócio sobre a qual a acção aplicacional foi executada. Esta é um conceito cuja relação não é obrigatória para com o evento, pois existem acções que são despoletadas nos serviços que não estão relacionadas com nenhuma entidade em específico, como por exemplo obter listagens de conteúdos.

Os argumentos consiste numa listagem que identifica todos os argumentos usados na acção, permitindo ao processamento analítico acesso a todos os parâmetros da operação e não só ao sujeito sobre quem foi feita a acção.

Estes dois conceitos serão caracterizados por duas propriedades que caracterizam a entidade de negócio: (i) O seu valor. (ii) O seu tipo.

Pretende-se que os valores usados na caracterização do sujeito sejam identificadores únicos no domínio aplicacional. Com a utilização dos identificadores únicos, torna possível ao processamento analítico o cálculo de métricas para KPIs relacionados com essa entidade de negócio, como por exemplo obter uma contagem para o número de vezes que um determinado TVOD foi alugado, ou uma contagem pelo número de pessoas que num determinado intervalo de tempo estavam sintonizados em determinado canal de televisão. A propriedade tipo terá de ser um valor que é transversal à organização, bem conhecido e definido no âmbito aplicacional do serviço onde o evento esteja a ser registado.

4.6.4 Domínio de equipamentos

O domínio de conhecimento de um equipamento é caracterizada por um identificador único que identifica o equipamento, bem como a caracterização de tipos de equipamento se trata. O departamento do produto define como equipamentos possíveis os apresentados na Tabela 16.

No caso de se tratarem de equipamentos que sejam propriedade do cliente, como por exemplo, *smartphones*, tablets ou mobile, o id do equipamento é gerado na aplicação e fica gravado na aplicação enquanto essa estiver instalada no dispositivo do cliente.

No caso de se tratar de equipamento do tipo web o identificador baseia-se num valor recolhido do *browser* que o utilizador tiver a usar.

Tabela 16 - Tipos de equipamento

Tipo	Descrição
SmartPhone	Equipamento móvel
Dongle	Dispositivo externo que se liga a um computador ou televisão
Tablet	Computador de espessura fina sem teclado
STB	*Set-top box*
Mobile	Equipamento móvel, como por exemplo um portátil
Web	Página Html com javascript associado em execução no *browser*
Server	Servidor do ecossistema.

No caso de se tratar de um servidor onde o serviço ou sistema esta em execução o identificador desse servidor será o seu nome.

4.6.5 Domínio do *software*

O domínio de conhecimento de *software* define três conceitos:

1. *Software*
2. Componente
3. Rotina.

O *Software* é um conceito definido pelo seu nome, e pela sua versão. O nome é um valor que identifica inequivocamente, e a versão permite saber as funcionalidades do *software*.

O componente é um conceito que define a entidade aplicacional onde o *log* foi registado, e é caracterizado por um nome e pelo tipo de componentes listados na Tabela 17.

A Rotina é um conceito também ele possível de se definir por um nome e pelo tipo apresentadas na Tabela 18.

Tabela 17 - Tipos de componente

Nome	Descrição
Classe	Identificação genérica.
Interface Aplicacional	Trata-se de um componente que recebe os pedidos das aplicações externas
Página Web	Página Web
Lógica	Componente que processa lógica aplicacional
Serviço	Componente que disponibiliza um serviço
Repositório	Componente que permite o acesso a uma base de dados

Tabela 18 - Tipos de rotina

Nome	Descrição
Método	Rotina de uma classe
Delegate	Rotina de um evento
HttpEndpoint	Rotina associada a um Uri HTTP
Construtor	Rotina associada à inicialização de um componente

4.6.6 Domínio de comunicação

O domínio de comunicação serve para caracterizar dados relacionados com a comunicação sobre IP. Este domínio é caracterizado por um IP e identificar a rede onde esse IP está inserido através de um identificador de rede.

Todos os sistemas do ecossistema da Figura 12 comunicam entre si por protocolos de comunicação aplicacionais que assentam sobre o protocolo IP da camada OSI[15], nomeadamente HTTP e AMQP. Por forma a simplificar este domínio de conhecimento a comunicação é definida pelos valores já definidos no *log* actual, nomeadamente o IP de quem originou o pedido e o identificador da rede onde esse cliente se situava. A origem do evento por sua vez tem uma relação com este domínio definindo assim a informação de rede da origem do evento.

4.6.7 Domínio de sessão

O domínio de sessão é composto por conceitos que caracterizam as sessões iniciadas pelo participante:

1. Identificação da sessão autenticada de utilizador
2. Identificação da sessão de *streaming* de vídeo.

Este domínio de conhecimento só está disponível quando o cliente estiver a consumir de uma plataforma OTT, por exemplo um equipamento móvel do cliente, caso contrário uma origem nunca vai ter uma relação com a sessão, uma vez que a comunicação entre a STB e os sistemas *FrontEnd Next Gen* acontecem dentro de uma rede privada da NOS.

4.6.8 Domínio do resultado

O domínio de conhecimento do resultado representa os conceitos associados à representação do resultado do evento aplicacional. Pela análise do *log* actual verificou-se a necessidade de associar ao final da ocorrência de um evento, informação do estado da operação e ainda a duração do evento. Algo que é possível introduzir neste modelo por forma a enriquece-lo é informação do domínio aplicacional do resultado da operação executada. Com a definição do domínio de conhecimento de domínios aplicacionais, é possível caracterizar de forma detalhada o objecto do resultado associando a ele o tipo de domínio aplicacional que é retornado. Esta informação é relevante para o processamento analítico pois permite caracterizar o resultado de uma operação que ocorreu parametrizada por um um conjunto de argumentos e sujeitos definidos pelo domínio de conhecimento de parâmetros.

Os domínios de conhecimento descritos nesta secção, foram possíveis de caracterizar através da análise do ecossistema de serviços, da análise do tipo de *software* que existe, da comunicação que os serviços e sistemas praticam entre si e quais os domínios aplicacionais existentes no ecossistema. Existe ainda um domínio de conhecimento que não foi caracterizado, o domínio de conhecimento da acção do evento. Para percebermos em que consiste a caracterização de uma acção de um evento serão apresentados casos de utilização de negócio que permitem perceber em que consiste este domínio de conhecimento.

[15] https://support.microsoft.com/en-hk/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained

4.6.9 Domínio de domínios aplicacionais

Cada um dos serviços apresentados na Figura 12, é responsável por um conjunto de conceitos de domínios aplicacionais, que acrescentam valor à ontologia deste trabalho porque tipificam os valores recolhidos. Um *log* com dados nesta natureza acrescenta valor ao processo analítico, porque permite:

1. Existência de tabelas de dimensões para descrever domínios de negócio, e manter actualizada a lista de entidades de negócio de forma automática. Quando surgir um novo conceito de negócio ele surgirá no processamento analítico através do registo de eventos, sem intervenção de pessoas a inserir esses registos, eliminando a duplicação de termos que representam conceitos existentes no negócio.
2. Reduzir a dependência para com importadores de bases dados aplicacionais. Estes são processos que consomem tempo mas relevantes, uma vez que é a partir de cópias de bases de dados que surge informação do consumo. No entanto, considera-se excessivo ter de depender deste mecanismo para poder saber a actividade de consumo. Esta ontologia iria permitir a criação de *logs* que remove essa necessidade ao processo analítico da organização.
3. Reduzir a volumetria de dados através da redução de processos de importação do ponto anterior. Reduzir o número de base de dados importados para saber o consumo dos clientes permite à organização permite usar essa cota de espaço para os *logs* dos eventos aplicacionais podendo influenciar favoravelmente os custos com o *datacenter*.
4. Permitir que o processamento analítico realize operações de cálculo de métricas de utilização das diferentes entidades de negócio em tempo real.
5. Homogeneidade na representação dos valores. Uma vez que todas as aplicações passam a gerar *logs* com base numa biblioteca representativa desta ontologia, tudo o que é enviado para processamento analítico independentemente do sistema representa o valor de *dominio* de negócio de forma igual.

Os conceitos de negócio que existem em cada um dos serviços do ecossistema da NOS Inovação encontram-se todos representados no sistema FrontEnd NextGen. Isto acontece porque este sistema tem a responsabilidade de orquestrar os casos de utilização das aplicações comunicando com todos os serviços de consumo. Os domínios de negócio que podem ser enquadrados em:

1. *Video on Demand* (VOD) – Conteúdos disponíveis sobre pedido do cliente
2. *Electronic Programming Grid (*EPG) – Conteúdos disponíveis ou relacionados com televisão
3. *Contents* – conceitos que modelam dados transversais a um dos dois anteriores
4. *User Interface* (UI) – conceitos específicos da *user interface* e da navegabilidade.
5. *Accounting* – Conceitos que modelam dados relacionados com a conta de cliente.

O Domínio aplicacional de VOD, é composto por conceitos que modelam a definição dos conteúdos que estão disponíveis ao cliente de serem consumidos quando o cliente queira, estão disponíveis a partir de duas aplicações: (i) videoclube (ii) NPlay. Estes conceitos podem ser encontrados na Tabela 19.

Tabela 19 - Conceitos VOD

Conceito	Descrição
VodCategory	Categoria agregadora de conteúdos VOD
Vod	Conteúdo de vídeo que está disponível para o cliente consumir
TVod	*Transactional VOD* – VOD que foi alugado
Svod	Conteúdo referente a uma subscrição de oferta comercial, ex: NPLAY
BVod	Grupos de VODs
PVod	Vod que foi comprado
Trailer	Resumo de um filme composto por algumas cenas do filme

O domínio aplicacional de EPG, é composto por indivíduos aplicacionais que caracterizam o tipo de dados relacionados com conteúdos da grelha televisiva. Como exemplo o canal de televisão um programa, um evento de um programa e ainda uma gravação de um programa. a lista completa de conceitos do domínio aplicacional encontram-se Tabela 20.

Tabela 20 - Conceitos EPG

Conceito	Descrição
EpgCategory	Categoria agregadora de conteúdos EPG
EPG	Grelha televisiva dos últimos 15 dias uteis
Programme	Programa de televisão que não está associado a uma serie
Channel	Canal de televisão
Season	Serie televisiva
Episode	Episódio de uma *Season*
Event	Ocorrência de um episódio de um programa num canal de televisão
PersonalRecording	Gravação do cliente associada ao identificador da conta de cliente
Schedule	Grelha televisiva de um canal dentro de uma janela temporal

A interface disponível ao utilizador também ela é modelada com um domínio próprio. Neste domínio aplicacional, definem-se conceitos usados na construção da UI. Estes conceitos, listados na Tabela 21 estão presentes no sistema *FrontEnd NextGen* uma vez que este serviço contém todas as regras de composição das secções da UI da aplicação cliente.

Tabela 21 - Conceitos de UI

Conceito	Descrição
Grid	Grelha de conteúdos que pode ser inserida em diferentes contextos
NodeItem	Item que contém um conteúdo para ser apresentado na UI
RootNodeItem	Nó de uma arvore de conteúdos que não tem pai associado mas pode ter um ou mais filhos
ChildNodeItem	Nó de uma arvore de conteúdos que tem um nó *NodeItem* como pai
Placeholder	Elemento da UI onde se colocam noteItems
Menuoption	Opção de um menu presente na UI

A categoria de *contents* aglomera todos os conceitos que estão presentes nos dois domínios apresentados anteriormente, EPG e VOD. Estes conceitos de alguma forma aparecem no sistema FrontEnd NextGen associados sempre a um dos dois conceitos anteriores.

Tabela 22 - Conceitos de contents

Conceito	Descrição
imdbRating	Rating extraído do IMDB
CastAndCrew	Categoria agregadora de pessoas que são as personagens o equipa responsável de um conteúdo VOD ou EPG
Like	Informação que o cliente gostou do conteúdo
Youtube	Conteúdos que são visualizados do youtube
Image	Representa uma imagem
Video	Representa um vídeo de um conteúdo VOD ou EPG
Tag	Descrição de característica do conteudo
Tagsgroup	Groupo de tags
Promo	Conteúdo promocional

Por último todos os conceitos que estão associado ao domínio de *Accounting*. *Accounting* engloba todos os conceitos do que está associado a um cliente enquanto entidade consumidora de conteúdos. Estes conceitos são usados dentro da organização para: (i) identificar um cliente (ii) descrever o seu portfólio de consumo (iii) equipamentos.

Tabela 23 - Conceitos de accounting.

Conceito	Descrição
Household	Conta de cliente que refere uma instalação de tv numa casa
Profile	Perfil do cliente associado ao OTT ou à STB
User	Identificação de uma pessoa com um login único que pode estar associado a uma *household*
Device	Equipamento físico de um cliente fornecido pela NOS associado ao contrato de TV, pode ser uma STB ou um *Router*.
Voiceline	Linha de voz que o cliente tem instalado em casa
Product	Producto comercial que o cliente adquiriu à NOS
Portfolio	Lista de produtos comerciais que caracteriza o consumo do cliente.

Estes são os conceitos que se encontram nos domínios aplicacionais dos ecossistemas, organizados por forma transversal a qualquer serviço ou sistema. A razão de organizar estes domínios desta forma é para abstrair os sistemas dos conceitos, apesar dos sistemas serem *owners* desses domínios aplicacionais estes são conceitos que estão presentes em mais do que um sistema, como é o caso do *FrontEnd NexGen* que utiliza todos estes conceitos. O conjunto composto por todos estes conceitos constituem o domínio de conhecimento de domínios aplicacionais da ontologia.

4.6.10 Domínio dos verbos das acções

Na execução de um caso de utilização, a comunicação entre os sistemas envolvidos, consiste num serviço a executar uma acção sobre um outro serviço por forma a alterar o estado de uma entidade de domínio, ou simplesmente obter informação sobre uma entidade ou uma lista de entidades. Esta comunicação é realizada sobre HTTP. Este protocolo disponibiliza semântica da acção pelos seus verbos e os usados actualmente na organização são:

1. GET – Obter recurso,
2. POST – Criar recurso,
3. PUT – Criar ou actualizar recurso,
4. DELETE – apagar recurso,
5. PATCH – actualizar um recurso parcialmente.

Apesar de ser um protocolo rico em descrição de acções, elas não são suficientes para descrever as acções aplicacionais do domínio do negócio de TV e OTT. Face a esta limitação o departamento do produto, especificou um conjunto de acções possíveis de ocorrerem sobre as entidades de domínio identificadas como parte do domínio de conhecimento de domínio aplicacional apresentado na secção 4.6.8. As entidades sobre as quais estão definidas acções são:

1. Vod

2. Svod
3. Bvod
4. Epg
5. personalRecording
6. youtubevideo.

4.6.10.1 Acções sobre VOD

Estas acções são comuns a todos os tipos de VODs.

Tabela 24 - Acções de VODs

Acção	Descrição
watch	Visualizar VOD
resume	Continuar a visual um conteúdo previamente pausado
watch_from_beginning	Visualizar VOD a partir do inicio
stop	Parar a visualização do VOD
watch_later	Agendar VOD para visualizar mais tarde
remove_watch_later	Remover VOD da lista de visualizações agendadas
language	Aceder às opções de língua do som do VOD
language_option	Alterar a Opção de língua do som do VOD
follow_series	Acompanhar alterações de catálogo de uma serie
unfollow_series	Parar de acompanhar alterações de uma serie
watch_trailer	Visualizar Trailer
go_to_main_asset	Navegar para o *content* pai de um SVOD
like	Marcar um gosto ao VOD
cancel_like	Cancelar a marcação de um gosto do VOD
dislike	Remover um gosto do VOD
cancel_dislike	Cancelar a remoção do gosto do VOD
mark_seen	Marcar o VOD como visualizado
mark_unseen	Marcar o VOD como não visualizado
remove_continue_watching	Remover o VOD da lista de VOD a continuar a ver
share	Partilhar um VOD
remote_watch_vod	Visualizar o VOD remotamente
remote_watch_vod_multi	Visualizar o VOD remotamente um vários ecrãs
remote_watch_trailer	Visualizar o Trailer do VOD
download	Fazer download de um VOD

4.6.10.2 Acções sobre TVOD

As acções que ocorrem sobre os TVODs consistem em acções que o utilizador pode fazer sobre conteúdos que estão associados ao cliente num período temporal reduzido, actualmente de 48h. Os TVODs são conteúdos que o cliente pode arrendar, pode comprar e sobre os quais pode aplicar a utilização de um *voucher*.

Tabela 25 - Acções de TVOD

Acção	Descrição
rent_tvod	Alugar um VOD
purchase_tvod	Comprar um TVOD
redeem_tvod_voucher	Activar código promocional sobre um TVOD
redeem_tvod_voucher_rental	Activar código promocional de aluguer de um TVOD
redeem_tvod_voucher_purchase	Activar código promocional de compra de um TVOD

4.6.10.3 Acções sobre SVOD

Tabela 26 - Acções de SVODs

Acção	Descrição
subscribe_svod	Subescrever um VOD
go_to_children_assets	Navegar pelos VODs da subscrição
redeem_svod_voucher	Activar código promocional sobre um SVOD
watch_trailer	Visualizar trailer de um VOD da subscrição
preview_all	Visualizar uma imagem representativa de todos

4.6.10.4 Acções de um BVOD

Tabela 27 - Acções de BVODs

Acção	Descrição
rent_bvod	Alugar um Bundle de VODs
purchase_bvod	Comprar um Bundle de VODs
redeem_bvod_voucher	Activar código promocional de um BVOD
redeem_bvod_voucher_rental	Activar código promocional de aluguer de um BVOD
redeem_bvod_voucher_purchase	Activar código promocional de compra de um BVOD
watch_all	Visualizar todos os VODS de um Bundle
watch_later	Marcar um BVOD para visualizar mais tarde
remove_watch_later	Remover da lista de visualizar mais tarde
watch_trailer	Visualizar trailer
preview_all	Visualizar uma imagem de todos os BVODs

4.6.10.5 Acções sobre EPG

Os conteúdos EPG agrupam-se em dois subdomínios:

1. Eventos de Programas – Visualização da transmissão de um programa. Esta transmissão pode acontecer em dois instantes temporais: (i) Visualizar o que está a ser transmitido naquele momento (ii) Visualizar o que foi transmitido algures no passado
2. Episódio de uma serie – Episodio de uma serie pode ser transmitido naquele momento ou no passado

Maior parte dos eventos já apresentados para os conteúdos VOD repetem-se para os conteúdos de EPG. No entanto existe um conjunto de acções que são especificas para o EPG apresentados na Tabela 28 e Tabela 29. O EPG tem conteúdos de dois tipos:

1. Programas isolados
2. Episódios de series

A visualização de um conteudo destes é baseado num evento que é criado. Apesar de serem dois conceitos diferentes, as acções que acontecem sobre cada um deles são bastante semelhantes como apresentado na Tabela 28 e Tabela 29.

Tabela 28 - Acções possíveis de executar sobre um evento isolado de EPG

Acção	Descrição
record_single	Gravar um evento de um programa
manage_single_recording	Gerir todas as gravações
delete_single_recording	Apagar uma gravação
protect_single_recording	Proteger a gravação de remoção
unprotect_single_recording	Remover a protecção da remoção da gravação
remote_startover	Recomeçar o programa que esta a dar naquele momento

Tabela 29 - Acções possíveis de executar sobre episódios de series

Acção	Descrição
record_series	Gravar uma serie
record_series_episode	Gravar um episódio de uma serie
record_series_season	Gravar uma temporada de uma serie
record_series_future_episodes	Gravar os episódios futuros
record_series_all_seasons	Gravar todas as temporadas
manage_series_recording	Gerir as gravações de uma serie
cancel_series_episode	Cancelar a gravação de um episódio de uma serie
cancel_series_season	Cancelar a gravação de uma temporada
cancel_series_future_episodes	Cancelar a gravação de episódios futuros
cancel_series_all_seasons	Cancelar a serie para todas as temporadas
delete_series_recording	Apagar uma gravação de uma serie
delete_series_episode	Apagar um episódio de uma serie
delete_series_season	Apagar a gravação de uma temporada de uma serie
delete_series_all_seasons	Apagar todas as temporadas de uma serie
protect_series_recording	Proteger contra remoção
unprotect_series_recording	Desproteger contra remoção

4.6.10.6 Verbos das acções

As acções são descritas por texto composto por um verbo que representa uma intenção de realizar uma acção e algumas definem ainda o tipo de conteúdos para o qual a acção esta associada, o que

se considera ser redundante nesta ontologia, uma vez que o conteúdo sobre o qual a acção está a ser executada é descrita pelo sujeito do domínio dos parâmetros. Removendo o termo que representa o conteúdo sobre o qual a acção pode ser executada, e reescrevendo o termo com *camel casing*, sintaxe definida na organização para a definição de valores com significado, ficamos com um conjunto de verbos que expressam a operação executada listados na Tabela 30.

Tabela 30 - Acções abstraídas do domínio aplicacional

Acção	Descrição
Watch	Visualizar
Resume	Continuar a ver
watchFromBeginning	Visualizar do inicio
Stop	Parar
watchlater	Adicionar à lista de ver mais tarde
RemoveWatchLater	Remover da lista de ver mais tarde
Language	Aceder à definição de língua do conteúdo
languageOption	Aceder opção de língua
Follow	Seguir alterações de um conteúdo
Unfollow	Deixar de Seguir alterações de um conteúdo
GoTo	Navegar entre conteúdos relacionados na UI
Like	Gostar de um conteúdo
CancelLike	Cancelar o Gostar de um conteúdo
Dislike	Remover o Gostar de um conteúdo
CancelDislike	Cancelar a remoção do Gostar
MarkSeen	Marcar conteúdo como visto
MarkUnseen	Marcar conteúdo como não visto
RemoveContinueWatching	Remover conteúdo da lista de continuar a ver
Share	Partilhar
RemoteWatch	Visualizar conteúdo remotamente
RemoteWatchMulti	Visualizar conteúdo remotamente em vários ecrãs
Download	Descarregar conteúdo para o dispositivo do cliente
Record	Gravar Conteúdo
Manage	Gerir informação do conteúdo
Delete	Apagar Conteúdo
Protect	Proteger conteúdo
UnProtect	Desproteger conteúdo
Cancel	Cancelar uma outra acção
Rent	Alugar
Purchase	Comprar
Redeem	Activar código
Subscribe	Subscrever
UnSubscribe	Terminar subscrição

A lista de acções definidas na Tabela 30, é composta por acções cujo nome é a composição de verbos definidos nessa listagem, isso significa, que existem acções realizadas sobre conteúdos que são exprimidas pela composição de um ou mais verbos.

Ao permitir que seja possível exprimir uma acção pela utilização de um ou mais verbos, permite que a criação de nomes de acções dos eventos seja um processo extensível. Mapear um para um um verbo para o nome de uma acção permite a reutilização de verbos para a definição de acções cuja definição é composta por mais do que um verbo.

Tabela 31 - Verbos extraidos das acções existentes no dominio aplicacional

Acção	Descrição
Watch	Visualizar
Resume	Continuar a ver
StartOver	Visualizar do início
Stop	Parar
watchLater	Adicionar à lista de ver mais tarde
Remove	Remover Conteudo
Language	Aceder à definição de língua do conteúdo
languageOption	Aceder opção de língua
Follow	Seguir alterações de um conteúdo
Unfollow	Deixar de Seguir alterações de um conteúdo
GoTo	Navegar entre conteúdos relacionados na UI
Like	Gostar de um conteúdo
Cancel	Cancelar o Gostar de um conteúdo
MarkSeen	Marcar conteúdo como visto
MarkUnseen	Marcar conteúdo como não visto
Share	Partilhar
Remote	Visualizar conteúdo remotamente
WatchMulti	Visualizar conteúdo remotamente em vários ecrãs
Download	Descarregar conteúdo para o dispositivo do cliente
Record	Gravar Conteúdo
Manage	Gerir informação do conteúdo
Delete	Apagar Conteúdo
Protect	Proteger conteúdo
Cancel	Cancelar uma outra acção
Rent	Alugar
Purchase	Comprar
Redeem	Activar código
Subscribe	Subscrever
Create	Criar Entidade
Update	Actualizar Entidade
Browse	Navegar numa categoria de conteúdos
List	Listar Conteúdos

Alguns exemplos da definição do nome da acção do evento composto por mais do que um verbo pode ser encontrado na Tabela 32. Este cenário apresenta a capacidade que este domínio de conhecimento dá ao registo do *log* ao definir verbos e não acções para representar o que aconteceu. Esta versatilidade no entanto adiciona responsabilidade a quem usa este domínio de conhecimento. A criação de nomes de acções com verbos que não façam sentido, ou a utilização de verbos redundantes no seu significado pode prejudicar o processo analítico para análise do que aconteceu.

Tabela 32 - Acções compostas por verbos

Acção	Verbos usados
UnProtect	Remove+Protect
UnSubscribe	Cancel+Subscribe
RemoteRecording	Remote+Recording
RemoteWatch	Remote+Watch
RemoteWatchMulti	Remote+WatchMulti
DisLike	Remove+Like
CancelLike	Cancel+Like
CancelDisLike	Cancel+Remove+Like
DeleteDownload	Delete+Download
RemoveWatchLater	Remove+WatchLater
RemotePurchase	Remote+Purchase
RemoveRent	Remote+Rent
RemoteRedeem	Remote+Redeem
RemoteStop	Remote+Stop

Com o domínio de conhecimento de eventos aplicacionais definido, fica definido todo o domínio de conhecimento da representação de eventos aplicacionais.

Esta secção termina a apresentação da fase um da solução deste trabalho. Numa abordagem bottom-up foi feito um estudo sobre três aspectos importantes que permitiram identificar o que é relevante para a conceptualização da representação do conhecimento de eventos aplicacionais.

A análise dos casos de utilização no processo analítico ganha mais valor do que o actual se o seu domínio de conhecimento for enriquecido com informação que descreva o ambiente onde o evento ocorreu e os seus intervenientes, informação que descreva referências temporais e o seu resultado. Com informação destes domínios concluiu-se que é possível analisar o sucesso ou insucesso de um caso de utilização e correlacionar um evento aplicacional de um caso de utilização com outros para gerar informação que caracterize um dos domínios de conhecimento relevantes. Através da análise do log actual identificou-se a informação que actualmente é recolhida e que é necessário continuar a enviar para processamento analítico, bem como a identificação de que é necessário alterar e melhorar no formato do log que é escrito. Esta alteração

terá de permitir que o processo CEP seja mais simples, menos extenso onde o formato do log recolhido terá de ser homogéneo e representado como um só, escrito para uma representação *standard*, possível de ser reutilizado e partilhado por vários sistemas, bem como entendido por pessoas bem como processos automáticos. Na secção 4.2 a análise do ecossistema de serviços da NOS Inovação permitiu ter uma visão sobre a quantidade de serviços existentes potenciais utilizadores de um log semântico. Para este trabalho pretende-se que a ontologia disponibilize um conjunto de indivíduos que descrevam os domínios aplicacionais existentes neste ecossistema, e ainda um conjunto de indivíduos que descrevam as acções possíveis de acontecer neste ecossistema. Na próxima secção é apresentada a segunda fase deste trabalho que consiste na construção e apresentação da ontologia que representa o domínio de conhecimento para descrever a ocorrência de eventos aplicacionais. Na segunda fase todo o estudo feito nesta primeira fase será revisitado e formalizado na definição da TBox da ontologia.

4.7 Definição da ontologia

Uma ontologia permite obter uma representação do conhecimento de determinado domínio, quer seja ele real, físico, ou virtual, onde se pretende representar a ocorrência de um evento aplicacional, num sistema ou aplicação com a especificação de conceitos do domínio de forma estruturada e com semântica associada [13, 6, 8, 15].

Uma ontologia consiste na existência de quatro componentes para representar um domínio, com os quais definimos a TBox e a ABox da ontologia.

Fazem parte da definição da TBox da ontologia:

1. Classes - Conceitos que representam um conjunto de entidades do domínio.
2. Propriedades – Características que fazem parte da definição de uma classe.
3. Taxonomia – Relações entre as classes.
4. Axiomas - Axiomas para descrever afirmações que são sempre verdade.

A definição de ABox da ontologia é composta por:

5. Indivíduos - Indivíduos que representam exemplos concretos de indivíduos do domínio.

Exprimir o significado da ocorrência de um evento, como identificado na secção 3.2.8 não é tarefa nova, já existem atualmente um conjunto de ontologias que abordam esta mesma problemática. No âmbito deste trabalho, alguns dos conceitos definidos dessas ontologias serão reutilizados, e outros serão criados de novo para descrever uma representação que englobe todos os conceitos de domínios de conhecimento identificados na primeira fase deste trabalho.

4.7.1 Requisitos não funcionais

A secção anterior definiu aquilo que a ontologia terá de exprimir através de um modelo comum. Nesta secção será descrito como é que este modelo terá de ser construído.

Como observado pela descrição do ecossistema da NOS Inovação na secção 4.2 e pelos casos de utilização na secção 0, existe mais do que um sistema envolvido no processo de caracterizar a ocorrência de um caso de utilização de negócio por parte de um cliente. A utilização do mesmo modelo por todos esses sistemas requere um conjunto de características não funcionais relevantes à utilização com sucesso da implementação desta ontologia. Os requisitos não funcionais relevantes que foram identificados semelhantes aos especificados na ontologia do modelo F são os apresentados na Tabela 33.

Tabela 33 - Requisitos não funcionais da ontologia

Requisito	**Descrição**
Axiomatização e precisão formal	É objetivo do modelo seja a base para um entendimento comum de como descrever eventos aplicacionais para garantir a interoperabilidade entre sistemas que utilizem esta descrição de eventos. É necessário que o suporte desta ontologia disponibilize mecanismos que automaticamente garanta a validade das relações das diferentes entidades recolhidas, dos seus valores e da sua semântica.
Reutilização	Reutilização do mesmo modelo por diferentes sistemas para caracterizar diferentes eventos, permite uma maior cobertura de sistemas onde seja possível utilizar os conceitos que definem a ocorrência de é um evento aplicacional.
Separação de responsabilidades	Permitir a integração e utilização do modelo em sistemas de diferentes natureza e de domínios de negócio diferentes. A responsabilidade desta ontologia é a de representar um evento aplicacional e essa representação é transversal a qualquer aplicação do ecossistema da NOS Inovação
Modularidade	cada uma das entidades que compõem um evento são recolhidas em diferentes momentos de um pedido HTTP onde está presente cada uma entidades identificadas nos requisitos funcionais. É importante que cada uma dessas representações de sub partes do evento seja usada independentemente das restantes, sendo todas agregadas antes do envio do evento para processamento analítico
Extensibilidade	O negocio de uma empresa evolui ao longo do tempo e por isso todos os seus sistemas acompanham essa mesma evolução. É necessário que esta ontologia suporte o aparecimento de novos conceitos de

	negócio e descreve-los nas várias componentes sem ser necessário alterar a sua definição.

4.7.2 Classes

A ontologia de eventos aplicacionais deste trabalho é definida pelo conjunto de todos os conceitos presentes nos domínios de conhecimento identificados:

1. Temporal
2. Participantes
3. Parâmetros
4. Equipamentos
5. *Software*
6. Comunicação
7. Sessão
8. Resultado.
9. Domínios aplicacionais.
10. Domínio de verbos das acções.

Cada um desses domínios define um ou mais conceitos, e baseados nesses conceitos serão definidas as classes desta ontologia definidas até agora.

Para expressar estes conceitos, a definição das classes e relações será apresentada de forma tabular com um breve resumo da sua definição seguido de uma listagem com código escrito sob um conjunto de tecnologias descritas para este fim, RDF, OWL escrito no no formato JSON-LD.

Em JSON-LD a definição de espaço de nomes deve ser realizada num objecto com o nome *@context.* A declaração dos espaços de nomes no contexto, vem permitir a utilização dos aliases por todo o documento melhorando assim a leitura desse mesmo documento.

Os espaços de nomes usados neste trabalho para a definição da nossa ontologia consiste na utilização de OWL, RDF e XML. Todas as classes e propriedades próprias desta ontologia estarão definidas sobre o espaço de nomes da ontologia designado de *Semantic structured events* - SSE.

```
"@context": {
      "sse" : "http://www.semanticweb.org/vitorpaulino/ontologies/2017/6/sse",
     "owl" :"http://www.w3.org/2002/07/owl",
     "rdfs" : "http://www.w3.org/2000/01/rdf-schema",
     "xmls" : "http://www.w3.org/2001/XMLSchema",
     “subclassof” : “owl#subClassOf”
  },
```

Listagem 10 - Definição dos aliases dos espaços de nomes usados na ontologia

4.7.2.1 Evento

A classe evento é a entidade que reúne todos os restantes conceitos através de um conjunto de relações de composição e especialização.

Tabela 34 - Definição da classe Event

Classe	Event
Definição	Um evento aplicacional é um acontecimento que ocorre dentro de uma janela temporal, numa rotina de código que recebe um conjunto de parâmetros e que produz um conjunto de valores.
Owl:subclassof	event:Event

```
{
"@id" : "sse#Event",
"@type" : [ "owl#Class" ]
}
```

Listagem 11 - Definição da classe Event em JSON-LD

4.7.2.2 Entidade

Durante a caracterização dos domínios de conhecimento identificados neste trabalho, muitos dos conceitos são caracterizados por dois atributos, nomeadamente o valor que identifica uma entidade que existe dentro de um determinado domínio de conhecimento e um segundo atributo que é o seu tipo. Os conceitos que têm vindo a ser caracterizados desta forma são:

1. Do domínio parâmetros: sujeito e argumentos
2. Do domínio *software*: componente e rotina

3. Do domínio de equipamentos: Equipamento
4. Do domínio de participantes: *User, Household, Profile*

A caracterização destes conceitos com estes dois atributos revela que todos estes conceitos se caracterizam da mesma forma, através de um identificador único e através de um descritor do dominio de negócio desse valor, a este conceito designaremos de Entidade.

Dada a definição da classe Entity da *ABC ontology* [32], este conceito enquandra-se nessa definição.

Tabela 35 - Classe Entity

Classe	Entity
Definição	Caracteriza uma entidade que pode pertencer a um dos domínios identificados previamente
Owl:subclassof	abc:Entity

```
{
  "@id" : "sse#Entity",
"@type" : [ "owl#Class" ],
“subclassof” : [“abc:Entity”]
}
```

Listagem 12 - Definição da classe Entity em JSON-LD

4.7.2.3 Temporal

O Domínio temporal é composto por 2 conceitos:

1. Instante temporal - Data e hora
2. Intervalo de tempo – Diferença de tempo entre dois instantes temporais

Estes conceitos podem-se encontrar já definidos na ontologia OWL *Time Ontology* [41] .Um instante temporal é definido pela classe *time:Instant*. Enquanto a duração é um conceito que pode ser definido como sendo do tipo *time:DateTimeInterval*.

Tabela 36 - Conceito Instante temporal

Classe	Instant
Definição	Uma entidade temporal sem duração associado
owl:equivalentClass	xsd:dateTime

Tabela 37 - conceitos de duração temporal

Classe	*TimeInterval*
Definição	Medição de tempo entre dois instantes temporais
owl:equivalentClass	*time:Duration*

4.7.2.4 Participante

Os conceitos definidos no domínio de conhecimento do participante são três, (i) *user, (ii) household* (iii) *Profile.* São entidades de negócio disjuntas cuja sua união caracteriza o participante envolvido. Cada um destes valores é representado pelo conceito Entidade e o seu conjunto forma o conceito participante.

Tabela 38 - Classe *Participant*

Classe	*Participant*
Definição	Identificação do cliente ou pessoa que participou num evento
Owl:subclassof	owl:thing

```
{
  "@id" : "sse#Participant",
"@type" : [ "owl#Class" ]
}
```

Listagem 13 . Definição da classe *Participant* em JSON-LD

Tabela 39 - Classe *User*

Classe	*User*
Definição	Identificação da pessoa que participou num evento
owl:subclassof	*Entity*

```
{
  "@id" : "sse#User",
  "@type" : [ "owl#Class" ],
  “subclassof” : [“sse#Entity”]
}
```

Listagem 14 - Definição da classe *User* em JSON-LD

Tabela 40 - Classe *Household*

Classe	*Household*
Definição	Identificação da conta de cliente que participou num evento
owl:subclassof	*Entity*

```
{
  "@id" : "sse#Household",
  "@type" : [ "owl#Class" ],
  “subclassof” : [“sse#Entity”]
}
```

Listagem 15 - Definição da classe *Household* em JSON-LD

Tabela 41 - Classe *Profile*

Classe	*Profile*
Definição	Identificação do perfil de cliente que participou num evento
owl:subclassof	*Entity*

```
{
  "@id" : "sse#Profile",
  "@type" : [ "owl#Class" ],
  “subclassof” : [“sse#Entity”]
}
```

Listagem 16 - Definição da classe Profile em JSON-LD

4.7.2.5 Parâmetros

O domínio de conhecimento parâmetros é definida por dois conceitos:

1. Sujeito
2. Parâmetros.

Ambos os conceitos têm como objetivo identificar e caracterizar entidades de domínio aplicacional. Para o processo analítico não existe necessidade de manter todos os atributos das entidades do domínio aplicacional para realizar interrogações, agregações ou correlações. Os atributos que se consideram relevantes são a sua identidade, através de um valor com o qual se identifique inequivocamente e a caracterização do seu domínio de negócio. Um outro atributo que estes conceitos usam refere-se à caracterização do domínio de negócio que consiste no conjunto de valores já identificados durante a caracterização do domínio de domínios aplicacionais na secção 4.6.8.

Tabela 42 - Classe Subject

Classe	*Subject*
Definição	Identificação da entidade sobre a qual ocorreu uma acção reportada pelo evento
owl:subclassof	*Entity*

```
{
  "@id" : "sse#Subject",
  "@type" : [ "owl#Class" ],
  “subclassof” : [“sse#Entity”]
}
```

Listagem 17 - Definição da classe *Subject* com JSON-LD

Tabela 43 - Classe *Parameter*

Classe	*Parameters*
Definição	Lista de valores que parametrizam a ocorrência da acção associada ao evento registado
owl:subclassof	*Entity*

```
{
  "@id" : "sse#Parameter",
  "@type" : [ "owl#Class" ],
  “subclassof” : [“sse#Entity”]
}
```

Listagem 18 - Definição da classe Parameter em JSON-LD

4.7.2.6 Equipamentos

O domínio de conhecimento de equipamento nesta ontologia é composto pelo conceito de equipamento já caracterizado na secção 4.6.4.

Tabela 44 - Classe *Device*

Classe	*Device*
Definição	Equipamento envolvido na ocorrência de um evento aplicacional
owl:subclassof	*Entity*

```
{
  "@id" : "sse#Device",
  "@type" : [ "owl#Class" ],
  “subclassof” : [“sse#Entity”]
}
```

Listagem 19 - Definição da classe *Device* em JSON-LD

4.7.2.7 Software

O domínio de conhecimento de *software* é composto por três conceitos:

1. *Software*
2. Componente
3. Rotina

Tabela 45 - *Classe Software*

Classe	Software
Definição	Identificação do software envolvido
Owl:subclassof	*owl:thing*

```
{
  "@id" : "sse#Software",
  "@type" : [ "owl#Class" ],
}
```

Listagem 20 - Definição da classe *Software*

Tabela 46 - Classe *Component*

Classe	*Component*
Definição	Identificação do componente onde a rotina executou
Owl:subclassof	*Entity*

```
{
  "@id" : "sse#Device",
  "@type" : [ "owl#Class" ],
  “subclassof” : [“sse#Entity”]
}
```

Listagem 21 - Definição da classe *Device* em JSON-LD

Tabela 47 - Classe *Routine*

Classe	*Routine*
Definição	Identificação do nome da rotina de código que executou
Owl:subclassof	*Entity*

```
{
  "@id" : "sse#Routine",
  "@type" : [ "owl#Class" ],
  “subclassof” : [“sse#Entity”]
}
```

Listagem 22 - Definição da classe Routine em JSON-LD

4.7.2.8 Comunicação

Para representar este domínio é criado o conceito *Network*. Este conceito é definido por todos os conceitos existentes que representam de forma coesa a definição de *Network*. Comunicação existente no ecossistema da NOS Inovação.

Tabela 48 - Classe *Network*

Classe	*Network*
Definição	Informação de comunicação na rede da NOS Inovação

Owl:subclassof	*Owl:thing*

```
{
  "@id" : "sse#Network",
  "@type" : [ "owl#Class" ],
}
```

Listagem 23 - Definição da classe *Network* em JSON-LD

4.7.2.9 Sessão

Semelhante à abordagem na definição de *Network*, a definição de *Session* segue a mesma abordagem. Tornar coesa a definição deste domínio de conhecimento consiste em definir *Session* como o conceito que é definido pelos conceitos existentes no domínio de conhecimento de sessão.

Tabela 49 - Classe *Session*

Classe	*Session*
Definição	Informação que caracteriza sessões que estão a decocorrer associadas ao participante do evento
Owl:subclassof	*Owl:thing*

```
{
  "@id" : "sse#Session",
  "@type" : [ "owl#Class" ],
}
```

Listagem 24 - Definição da classe *Session* em JSON-LD

4.7.2.10 Resultado

O resultado de um evento é também ele um conceito que pode ser modular como os anteriores, sendo definido como a composição de entidades definidas pelos conceitos deste dominio de conhecimento descrito na secção 4.6.8. Definindo uma classe que represente este domínio de conhecimento, é possível tratar o resultado de um evento e cruza-lo com outros eventos de forma desacoplada e coesa.

Tabela 50 - Classe *Result*

Classe	*Result*

Definição	Informação que caracteriza o resultado de uma acção associada a um evento
Owl:subclassof	*Owl:thing*

```
{
  "@id" : "sse#Result",
  "@type" : [ "owl#Class" ],
}
```

Listagem 25 - Definição da classe *Result* em JSON-LD

4.7.2.11 Domínios aplicacionais

A criação de uma classe que represente os domínios aplicacionais foi simplificada à simples descrição textual do nome do domínio de negócio. Este domínio de conhecimento é composto pelo conceito de *Domains* que descreve um conjunto de indivíduos cada um por cada conceito de domínio aplicacional listado na secção 4.6.9.

Tabela 51 - Classe *Domains*

Classe	*Domains*
Definição	Descrição de domínios aplicacionais
Owl:subclassof	*Owl:thing*

```
{
  "@id" : "sse#Domains",
  "@type" : [ "owl#Class" ],
}
```

Listagem 26 - Definição da classe *domains* em JSON-LD

4.7.2.12 Verbos de acções

A conceptualização da representação dos verbos das acções, foi realizado da mesma forma que os domínios aplicacionais. Este domínio de conhecimento é composto por um conceito de nome *EventAction* que permite definir indivíduos em que cada um corresponde a um verbo apresentado na secção 4.6.10.

Tabela 52 - Definição da classe *EventActions*

Classe	*EventActions*
Definição	Descrição de verbos aplicacionais
Owl:subclassof	*Owl:thing*

```
{
  "@id" : "sse#EventActions",
  "@type" : [ "owl#Class" ],
}
```

Listagem 27 - Definição da classe EventActions em JSON-LD

Este conjunto de classes definem os conceitos definidos nos vários domínios de conhecimento identificados durante a primeira fase deste trabalho.

4.7.3 Taxonomia

Existem dois tipos de relações entre as classes definidas numa ontologia utilizadas neste trabalho. Relações taxonómicas, através das quais se apresentam as relações das generalizações entre entidades, e relações entre classes através da definição de propriedades.

A relação taxonómica desta ontologia envolve a *Event ontology*, *ABC ontology*, e o conceito *Thing* definido em OWL. As relações taxonómicas que relacionam as entidades desta ontologia com as entidades fora do domínio desta ontologia são representadas a traço descontínuo, enquanto as relações de generalização entre entidades da ontologia são representadas por traço continuo. O sentido da generalização é descrito em todas as relações no sentido de baixo para cima.

4.7.3.1 Generalização Entity

A classe *User*, *Profile* ou *Household* são especialização da generalização Entity que por sua vez é uma especialização de Entity da *ABC Ontology*. A escolha desta ontologia em vez do conceito de Entity da ontologia DOLCE+DnS Ultralite, este relacionado com o facto da definição da DUL:Entity apresentada na Listagem 28, é uma classe que é definida pela união de um conjunto de classes que não têm domínio de conhecimento associado no âmbito desta organização.

```
<owl:Class rdf:about="http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Entity">
<rdfs:label xml:lang="en">Entity</rdfs:label>
<rdfs:label xml:lang="it">Entità</rdfs:label>
<owl:equivalentClass>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<rdf:Description rdf:about=
"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Abstract"/>
<rdf:Description rdf:about=
"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Event"/>
<rdf:Description rdf:about=
"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Object"/>
<rdf:Description rdf:about=
"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Quality"/>
<rdf:Description rdf:about=
"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Region"/>
</owl:unionOf>
</owl:Class>
</owl:equivalentClass>

<rdfs:isDefinedBy rdf:resource="http://www.ontologydesignpatterns.org/ont/dul/DUL.owl
"/>
</owl:Class>
```

Listagem 28 - DUL:Entity [55]

As classes *Object*, *Quality* e o conceito *Region* são conceitos demasiado abrangentes para a definição de entidade deste trabalho.

A classe Object é uma entidade que representa um objecto semelhante a um objecto físico ou social, físico, imaginário, que está sempre presente num evento. Indivíduos desta classe estão sempre associados a um local. Neste trabalho não existe o conceito de uma *Entity* física, uma vez que representam entidades virtuais que são representações de conteúdos que estão guardados numa base de dados aplicacional. Neste trabalho o conceito mais semelhante que existe com o local é a entidade *Device* que identifica e descreve os equipamentos, mas que não são representados ela sua localização, mas sim pelo seu nome. Ou identificador.

No entanto a classe abc:Entity consiste no que está definido na Listagem 29. O seu significado é suficientemente generalista mas ao mesmo tempo abrange o que se pretende para este conceito.

```
<rdfs:Class rdf:ID="Entity"/>
```

Listagem 29 - ABC:Entity [32]

4.7.3.2 Generalização owl:Thing

As especializações de *owl:thing* apresentadas na Figura 22 pela sua definição poderiam estar relacionados taxonomicamente com classes da ontologia DOLCE+DnS Ultralite, ou OpenCyc, mas os atributos e relações que iriam herdar iria alargar o âmbito da definição desses conceitos que no âmbito da organização considera-se desnecessário pois iriam importar mas conceitos a gerir que não têm significado nem existem na organização.

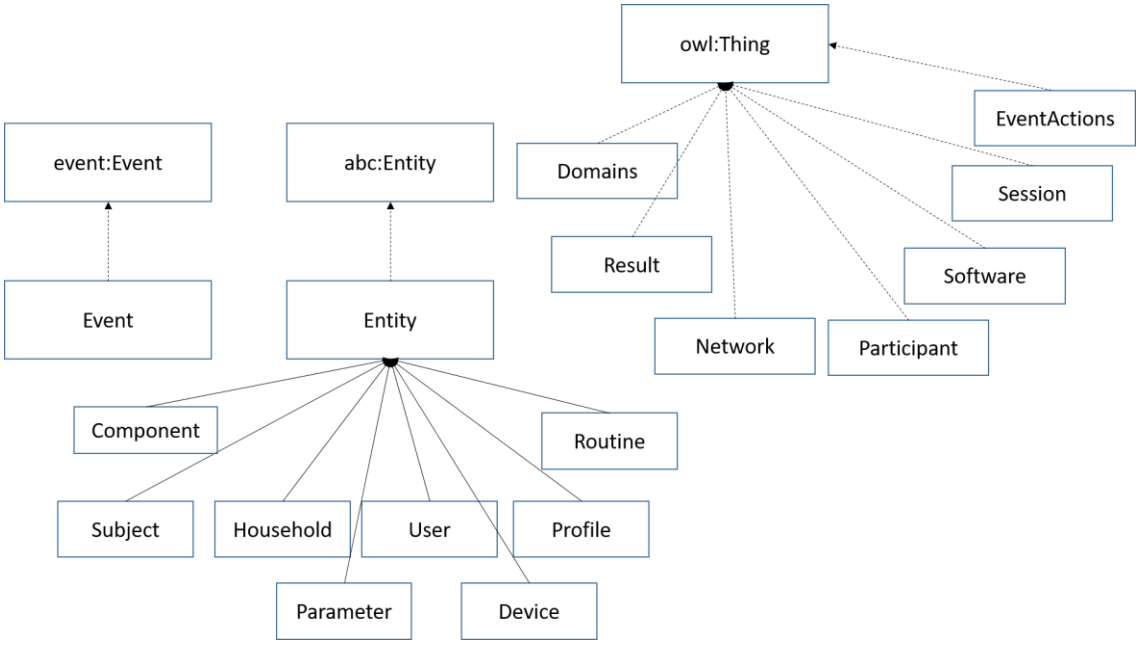

Figura 22 - Taxonomia da ontologia

4.7.3.3 Generalização de Event

A entidade *Event* desta ontologia representa a ocorrência de um acontecimento que mais se assemelha com a definição da *Event Ontology* descrita na secção 3.2.8. Apesar de *Event* estar definido em ontologias de alto nível, ontologias cujos conceitos são abrangentes o suficientes para qualquer domínio, a sua definição engloba conceitos que não existem no âmbito da organização ou do objectivo desta ontologia de domínio, como é o caso da ontologia DUL. A definição de *Event* na ontologia ModelF que é herdada da ontologia DUL é caracterizada como sendo uma sub classe de *DUL:Entity*. Um *Event* Sendo uma subclasse de *DUL:Entity* é caracterizado por todas as características que DUL:Entity. Esta entidade por sua vez é uma classe equivalente à união de um conjunto de classes, tais como:

1. Dul:Abstract – Entidades que não estão localizadas nem no espaço nem no tempo
2. Dul:Object – Um objecto pode ser um objecto físico ou social
3. Dul:Quality – Um aspecto de uma entidade que não existe sem estar associado a uma entidade
4. Dul:Region - Representa uma região georeferenciada

A presença de conceitos que não fazem parte do domínio da organização é algo que se evita, uma vez que o aparecimento de conceitos é algo que tem de estar associado a requisitos funcionais identificados na organização.

4.7.4 Propriedades

Da análise realizada na primeira fase deste trabalho, existem relações entre domínios de conhecimento que estavam implicitamente expressas pela definição do seu valor. O *log* actual

representa o tipo de dados que têm de se continuar a enviar. Para manter o que era registado com o *log* actual, Um evento aplicacional apresenta no seu domínio as seguintes propriedades:

1. CreatedAt – Quando é que aconteceu.
2. hasParticipant - Quem esteve envolvido.
3. hasOrigin - Quem causou o evento.
4. hasTarget - Onde o evento ocorreu.

Da análise feita ao ecossistema da NOS e aos casos de utilização verificamos que este *log* pode ser enriquecido com informação que:

1. hasSubjects - Descreve o sujeito sobre o qual a acção foi feita.
2. hasArguments - Descreve os argumentos da rotina de código.
3. hasResult - Descreve o resultado da acção.
4. hasActionName - Nome da acção.

O mapa de propriedades do evento consiste no conjunto de relações apresentadas na Figura 23.

A classe Participante tem relações para as entidades *User, Profile e Household* com uma cardinalidade de 1 cada uma.

A classe Caller reúne um conjunto de relações com classes que definem de onde foi feita a chamada. Caller relaciona-se com *Device, Software, Session* todas estas com propriedades do Tipo owl:ObjectProperty e ainda a língua, uma propriedade do tipo *owl:dataTypeProperty*.

A classe Callee reúne um conjunto de relações com as classes que definem onde ocorre o evento, nomeadamente com a classe *Software, Component, Routine e Device* todas elas do tipo *owl: ObjectProperty*.

A classe Result tem associada a si propriedades que caracterizam o resultado da acção, o que a acção gerou a data e hora do fim da acção e a duração da acção associada ao evento. Todas estas relações têm a restrição de cardinalidade de no mínimo 0 e no máximo 1.

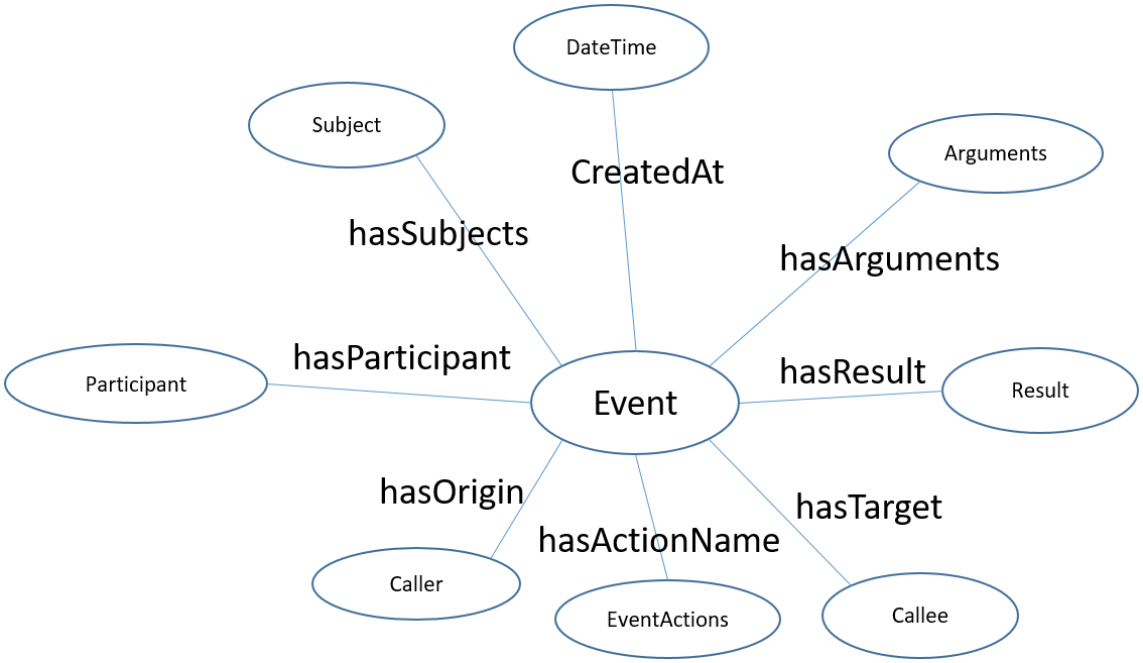

Figura 23 - Propriedades da classe *Event*

Na Figura 23 aparecem duas entidades que ainda não foram descritas. A palavra *Caller*, é reutilizada do domínio de desenvolvimento de *software*. Designa-se por *Caller*[16] a entidade que faz a invocação a uma rotina de código. A palavra *Callee*[17] tem a mesma origem, e serve para caracterizar a entidade ou rotina de código que foi invocada por outra rotina de código.

A entidade *Caller* e *Callee* são entidades que modelam relações com conceitos já descritos anteriormente:

1. *Caller* – Descreve relações que representam qual foi o *software* a partir do qual ocorreu a invocação, que comunicação realizou, qual a sessão de cliente que estava a decorrer naquele momento e qual a cultura que estava configurada no cliente.
2. *Callee* – Descreve relações que representam em que software é que o evento foi registado, em que componente desse componente e rotina de código e ainda qual o equipamento onde esse *software* estava em execução.

Semelhante à abordagem seguida na apresentação das classes, também a descrição das propriedades da ontologia terão a mesma estrutura. Cada uma das propriedades será descrita pelo seu significado e pela sua definição escrita em JSON-LD.

[16] https://en.wiktionary.org/wiki/caller
[17] https://en.wiktionary.org/wiki/callee

4.7.4.1 hasActionName

Uma relação entre *Event* e um individuo da classe *EventActions* que resulta em descrever a acção de um evento.

Tabela 53 - Descrição da propriedade *hasActionName*

Propriedade	*hasActionName*
Definição	Nome da acção executada
Tipo de propriedade	*owl:ObjectProperty*
Rdf:range	*Sse:EventActions*
Rdf:Domain	*Sse:Event*

```
{
          "@id" : "sse#hasAction",
          "@type" : [ "owl#ObjectProperty" ],
          "rdf#domain" : [ {
            "@id" : "sse#Event"
          } ],
          "rdf#range" : [ {
            "@id" : "sse#EventActions"
          } ]
        }
```

Listagem 30 - Definição da propriedade hasActionName em JSON-LD

Associada a esta relação existe uma restrição, nomeadamente ao número de valores mínimos que têm de existir. Um evento pode descrever uma acção com um ou mais verbos.

```
{
          "@id" : "sse#Action",
          "@type" : [ "owl#Class" ],
          "owl#subClassOf" : [ {
            "owl:Restriction" : {
                    “owl:onProperty” : “sse:hasActionName”,
                   “owl:minCardinality”:
                    {
                        “@id” : “1”,
                        “rdf:datatype” : “xsd:nonNegativeInteger”
                    }
            }
          } ],
```

Listagem 31 - Definição da restrição de cardinalidade da propriedade hasActionName em JSON-LD

4.7.4.2 hasResult

Relação que existe entre o evento e o resultado da acção que o evento descreve.

Tabela 54 - Definição da propriedade hasResult

Propriedade	*hasResult*
Definição	Resultado da acção que o evento descreve
Tipo de propriedade	*owl:ObjectProperty*
Rdf:range	*Sse:Result*
Rdf:Domain	*Sse:Event*

```
{
          "@id" : "sse#hasResult",
          "@type" : [ "owl#ObjectProperty" ],
          "rdf#domain" : [ {
            "@id" : "sse#Event"
          } ],
          "rdf#range" : [ {
            "@id" : "sse#Result"
          } ]
        }
```

Listagem 32 - Definição da propriedade hasResult em JSON-LD

Associada a esta relação existe uma restrição relativa à cardinalidade de resultados que um evento pode descrever. Como um evento tem uma relação de 1-1 com a acção que está a descrever, o resultado também é só um.

```
{
          "@id" : "sse#Result",
          "@type" : [ "owl#Class" ],
          "owl#subClassOf" : [ {
            "owl:Restriction" : {
                   “owl:onProperty” : “sse:hasResult”,
                   “owl:maxCardinality”:
                    {
                        “@id” : “1”,
                        “rdf:datatype” : “xsd:nonNegativeInteger”
                    }
            } } ],
```

Listagem 33 - Definição da restrição de cardinalidade de hasResult em JSON-LD

4.7.4.3 hasArguments

A propriedade hasArguments descreve a relação que os argumentos de uma acção têm com o evento que descreve a ocorrência da acção.

Tabela 55 - Definição da propriedade hasArguments

Propriedade	*hasArguments*
Definição	Relação relação que os argumentos de uma acção têm com o evento que descreve a ocorrência da acção
Tipo de propriedade	*owl:ObjectProperty*
Rdf:range	*Sse:Arguments*
Rdf:Domain	*Sse:Event*

```
{
          "@id" : "sse#hasParameter",
          "@type" : [ "owl#ObjectProperty" ],
          "rdf#domain" : [ {
            "@id" : "sse#Event"
          } ],
          "rdf#range" : [ {
            "@id" : "sse#Parameter"
          } ]
        }
```

Listagem 34 - Definição da propriedade hasArguments em JSON-LD

A existência de parâmetros numa acção é opcional e portanto a existência desta relação é opcional, descrita pela cardinalidade da Listagem 35.

```
{
          "@id" : "sse#Parameter",
          "@type" : [ "owl#Class" ],
          "owl#subClassOf" : [ {
            "owl:Restriction" : {
                   “owl:onProperty” : “sse:hasParameter”,
                   “owl:minCardinality”:
                    {
                        “@id” : “0”,
                        “rdf:datatype” : “xsd:nonNegativeInteger”
                    }
            } } ],
```

Listagem 35 - Restrição de cardinalidade da propriedade hasParameters em JSON-LD

4.7.4.4 hasSubjects

A propriedade *hasSubject* descreve a relação que os sujeitos de uma acção têm com o evento que descreve essa acção.

Tabela 56 - Definição da propriedade *hasSubjects*

Propriedade	*hasSubjects*
Definição	Relação que os sujeitos de uma acção têm com o evento que descreve essa acção
Tipo de propriedade	*owl:ObjectProperty*
Rdf:range	*sse:subject*
Rdf:Domain	*sse:Event*

```
{
          "@id" : "sse#hasSubjects",
          "@type" : [ "owl#ObjectProperty" ],
          "rdf#domain" : [ {
            "@id" : "sse#Event"
          } ],
          "rdf#range" : [ {
            "@id" : "sse#Subject"
          } ]
        }
```

Listagem 36 - Definição da propriedade *hasSubjects* em JSON-LD

Uma acção pode ter mais do que um sujeito associado, tipicamente quando se pretende realizar a mesma acção sobre um conjunto de entidades de negócio. Como por exemplo validar se um cliente tem autorização em visualizar uma lista de canais.

```
{
          "@id" : "sse#Subject",
          "@type" : [ "owl#Class" ],
          "owl#subClassOf" : [ {
            "owl:Restriction" : {
                  “owl:onProperty” : “sse:hasSubjects”,
                  “owl:minCardinality”:
                   {
                       “@id” : “0”,
                       “rdf:datatype” : “xsd:nonNegativeInteger”
                   }
            } } ],
```

Listagem 37 - Definição de restrição de cardinalidade da propriedade *hasSubjects* em JSON-LD

4.7.4.5 hasTarget

Esta propriedade descreve a relação que existe entre um evento e a classe que descreve o local onde o evento ocorreu.

Tabela 57 - Definição da propriedade hasTarget

Propriedade	*hasTarget*
Definição	Relação que existe entre o local onde a acção ocorreu e o evento que descreve esse acontecimento
Tipo de propriedade	*owl:ObjectProperty*
Rdf:range	*sse:Callee*
Rdf:Domain	*sse:Event*

```
{
          "@id" : "sse#hasTarget",
          "@type" : [ "owl#ObjectProperty" ],
          "rdf#domain" : [ {
            "@id" : "sse#Event"
          } ],
          "rdf#range" : [ {
            "@id" : "sse#Callee"
          } ]
        }
```

Listagem 38 - Definição da propriedade hasTarget em JSON-LD

Um evento representa a ocorrência de uma unica acção que acontece num sistema.

```
{
          "@id" : "sse#Callee",
          "@type" : [ "owl#Class" ],
          "owl#subClassOf" : [ {
            "owl:Restriction" : {
                   “owl:onProperty” : “sse:hasTarget”,
                   “owl:Cardinality”: {
                        “@id” : “1”,
                        “rdf:datatype” : “xsd:nonNegativeInteger”
                    }
            } } ],
```

Listagem 39 - Definição de restrição de cardinalidade da propriedade hasTarget em JSON-LD

4.7.4.6 hasOrigin

A origem de um evento caracteriza a informação do local onde ocorreu a invocação que originou o registo do evento.

Tabela 58 - Definição da propriedade *hasOrigin*

Propriedade	*hasOrigin*
Definição	Relação que existe entre o local onde foi realizada a invocação do sistema onde o evento foi registado.
Tipo de propriedade	*owl:ObjectProperty*
Rdf:range	*sse:Caller*
Rdf:Domain	*sse:Event*

```
{
          "@id" : "sse#hasOrigin",
          "@type" : [ "owl#ObjectProperty" ],
          "rdf#domain" : [ {
            "@id" : "sse#Event"
          } ],
          "rdf#range" : [ {
            "@id" : "sse#Caller"
          } ]
        }
```

Listagem 40 - Definição da propriedade *hasOrigin* em JSON-LD

Um acontecimento tem como causalidade apenas um local. Um acontecimento quando ocorre num sistema deveu-se a que só um outro sistema fez uma invocação sobre o sistema actual.

```
{
          "@id" : "sse#Caller",
          "@type" : [ "owl#Class" ],
          "owl#subClassOf" : [ {
            "owl:Restriction" : {
                   “owl:onProperty” : “sse:hasOrigin”,
                   “owl:Cardinality”: {
                        “@id” : “1”,
                        “rdf:datatype” : “xsd:nonNegativeInteger”
                    }
```

Listagem 41 - Definição da restrição de cardinalidade de *Caller* no evento.

4.7.4.7 hasParticipant

Um interveniente é caraterizado por diferentes valores, *User*, *Profile*, *Household* que representam um cliente.

Tabela 59 - Definição da propriedade *hasParticipant*

Propriedade	*hasIntervenient*
Definição	Um interveniente associa o cliente ao acontecimento.
Tipo de propriedade	*owl:ObjectProperty*
Rdf:range	*sse:Participant*
Rdf:Domain	*sse:Event*

```
{
          "@id" : "sse#hasParticipant",
          "@type" : [ "owl#ObjectProperty" ],
          "rdf#domain" : [ {
            "@id" : "sse#Event"
          } ],
          "rdf#range" : [ {
            "@id" : "sse#Participant"
          } ]
        }
```

Listagem 42 - Definição da propriedade hasParticipant

Um evento ocorre sempre associado a um só cliente.

```
{
          "@id" : "sse#Participant",
          "@type" : [ "owl#Class" ],
          "owl#subClassOf" : [ {
            "owl:Restriction" : {
                  “owl:onProperty” : “sse:hasParticipant”,
                  “owl:Cardinality”: {
                       “@id” : “1”,
                       “rdf:datatype” : “xsd:nonNegativeInteger”
                   }
            } } ],
```

Listagem 44 - Definição da cardinalidade de um participante num evento

4.7.4.8 CreatedAt

Um evento ocorre sempre com uma data e hora associada. Esta propriedade caracteriza a entidade temporal que regista quando é que a acção começou a executar, mas ainda não terminou a sua execução.

Tabela 60 - Definição da propriedade *CreatedAt*

Propriedade	*CreatedAt*
Definição	Relação que existe entre o evento e um instante temporal.
Tipo de propriedade	*owl:ObjectProperty*
Rdf:range	*xmls:DateTime*
Rdf:Domain	*sse:Event*

```
{
          "@id" : "sse#Created",
          "@type" : [ "owl#DatatypeProperty" ],
          "rdf#domain" : [ {
            "@id" : "sse#Event"
          } ],
          "rdf#range" : [ {
            "@id" : "xmlns:DateTime"
          } ]
        }
```

Listagem 45 - Definição da propriedade CreateAt em JSON-LD

Um evento quando ocorre fica associado a apenas uma data e hora de criação. Um evento é uma entidade que não é criado mais do que uma vez e depois terminar a sua execução não é alterado, sendo por isso uma entidade imutável.

A definição completa da ontologia pode ser encontrada no Anexo A, onde estão definidas as relações de cada uma das classes. A Figura 24 apresenta uma visão em grafo das várias classes e das relações que existem entre as várias classes. Através deste grafo consegue-se verificar que a ontologia consiste num conjunto de módulos que para os vários domínios de conhecimento apresentados na secção 4.6. A abordagem *bottom-up* permitiu através da análise do que existe hoje, extrair um domínio de conhecimento com o qual se conseguisse caracterizar um acontecimento numa aplicação. Os casos de utilização permitiram perceber qual a dinâmica que existe no ecossistema da NOS e permitiu identificar um conjunto de requisitos, nomeadamente:

1. Extrair informação que fosse relevante para análise de KPIs
2. Extrair informação que fosse relevante para análise de monitoria.

O estudo de um conjunto de casos de utilização existentes, permitiu identificar que nunca existe um sistema sozinho envolvido a dar suporte a um caso de utilização e que para validar que um caso de utilização acontece com sucesso todos os sistemas envolvidos têm de registar um evento aplicacional que regista o acontecimento de uma acção no seu sistema, enriquecendo assim o processamento analítico com o detalhe de cada um dos sistemas envolvidos que como um todo.

Através da análise do *log* actual, identificou-se os requisitos mínimos que a ontologia representa:

1. A Identificação do instante temporal de quando é que aconteceu
2. Caracterização do cliente
3. Caracterização de informação de rede onde o equipamento do cliente se encontra, nomeadamente a STB,
4. Equipamento do cliente
5. Estado do resultado da acção
6. Parâmetros da acção

Através da análise das tabelas existentes no processo analítico identificou-se qual a tarefa principal que é executada, nomeadamente identificar as acções que o cliente executa. Outro dos dados que era mantido estava relacionado com dados extraídos de uma segunda fonte de *logs* – *Logs do IIS*. Para remover essa dependência acrescentou-se também esses campos à definição do que o evento teria de representar, nomeadamente o nome da acção, o estado de sucesso ou insucesso da acção e a informação de caracterizar a aplicação cliente. A ontologia definida resultou ficou assim definida como sendo uma representação de conhecimento que permite responder as questões d'O que é que aconteceu?, Quando é que aconteceu? Quem é que fez acontecer? Onde é que aconteceu? Quem é que participou? Qual foi o resultado?.

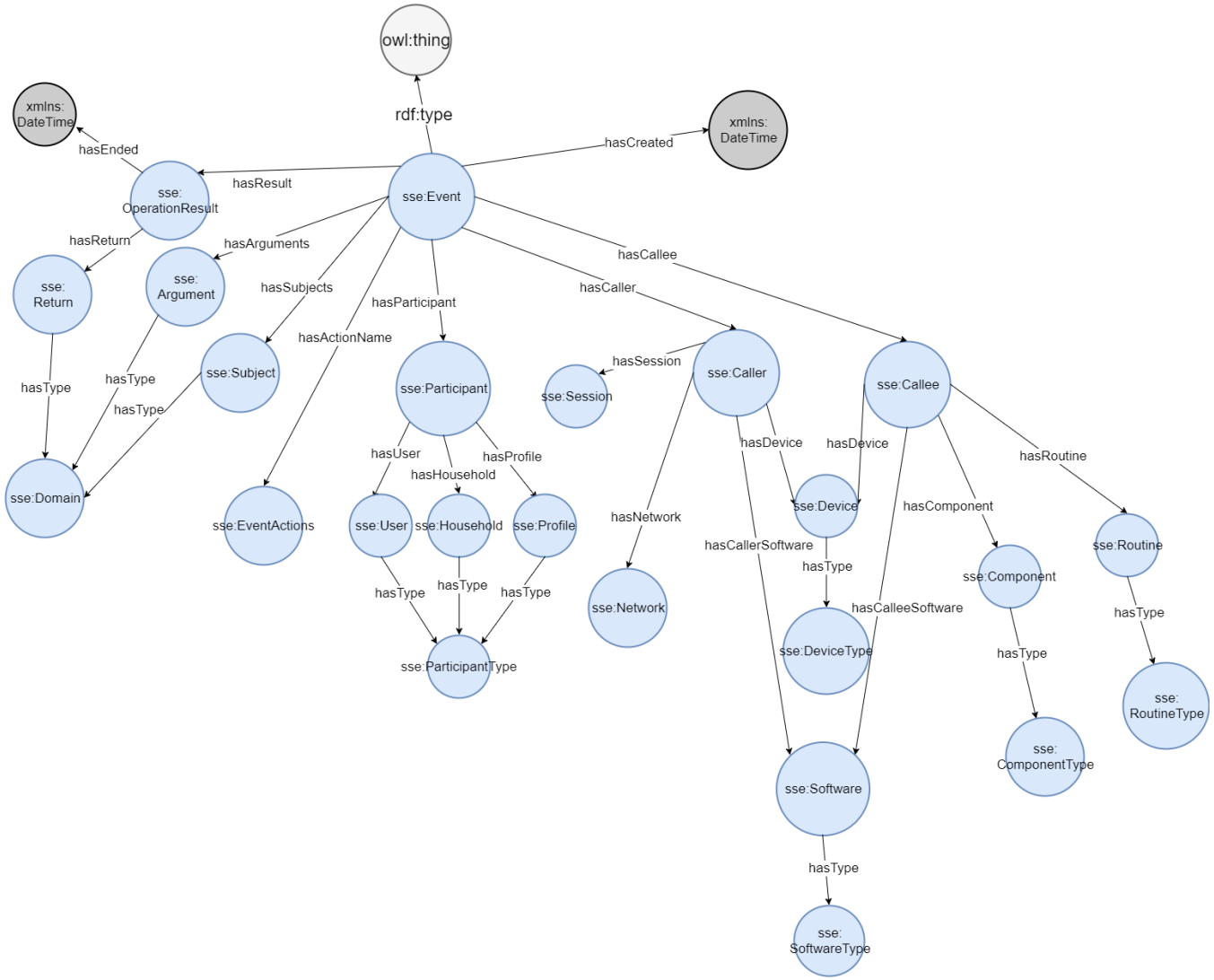

Figura 24 - Grafo da ontologia SSE

5 Solução proposta - Biblioteca e avaliação

5.1 Biblioteca aplicacional

A apresentação da implementação da biblioteca de código neste trabalho consiste na última fase deste trabalho. O âmbito desta fase, é o de evidenciar a utilização da ontologia definida na secção 4.7. Esta ontologia é definida por um conjunto de classes e propriedades que são o vocabulário que se pretenda que seja transversal, por todo o ecossistema de serviços da empresa no âmbito de registo de acontecimentos relevantes nas aplicações, como é o caso em estudo dos principais casos de utilização disponíveis aos utilizadores.

O âmbito desta biblioteca de código, não só consiste em completar a definição da base de conhecimento da nossa ontologia, através da instanciação de asserções da TBox resultando na ABox da ontologia, mas também de toda a infraestrutura que apoia ao processo. Esta biblioteca deverá disponibilizar componentes para detetar o início de um evento, num sistema, criar o evento, preenche-lo com os valores correctos, construir uma representação num formato que seja transportável e possível de leitura por parte de processos automáticos, mas também por seres humanos até ao último passo para o enviar para processamento analítico. Os principais requisitos identificados na organização que esta biblioteca deverá suportar são:

1. Modular
2. Coesa
3. Coerente
4. Suporte a IoC (*Inversion of control*)
5. API simples e familiar de se utilizar
6. Envio do log por múltiplos protocolos
7. Possibilidade de reportar no log todos os parâmetros e resultado de uma acção
8. O menor número possível de locais onde seja necessário escrever código
9. Suporte nativo para JSON para extensível para outros formatos.
10. Poder usar em aplicações e sistemas de diferentes plataformas

O registo da ocorrência desses eventos consistirá num conjunto de operações apresentadas na Figura 25, Registar, Normalizar, Serializar e por último Publicar. O registo da ocorrência de um evento será realizado nos serviços gestores de consumo da Figura 12.

Toda a implementação desta biblioteca segue as boas práticas de desenvolvimento de *software* orientado a objectos, designada por SOLID[18]. A utilização dessa API permite desacoplar e tornar coeso e coerente o processo de registo dos valores da implementação de negócio dos serviços.

Os dados recolhidos do que aconteceu, sobre o que aconteceu e qual o resultado desse acontecimento e quais os parâmetros desse acontecimento são normalizados para um vocabulário próprio para este efeito apresentado no capítulo 4. Após essa recolha de dados normalizada ao

[18] https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

vocabulário da ontologia, esses dados serão serializados para um formato baseado em JSON, sobre o qual seja possível realizar processos CEP [3]. Depois de serializado, será enviado em tempo real para processamento analítico por um dos meios de comunicação possíveis que esta biblioteca disponibiliza.

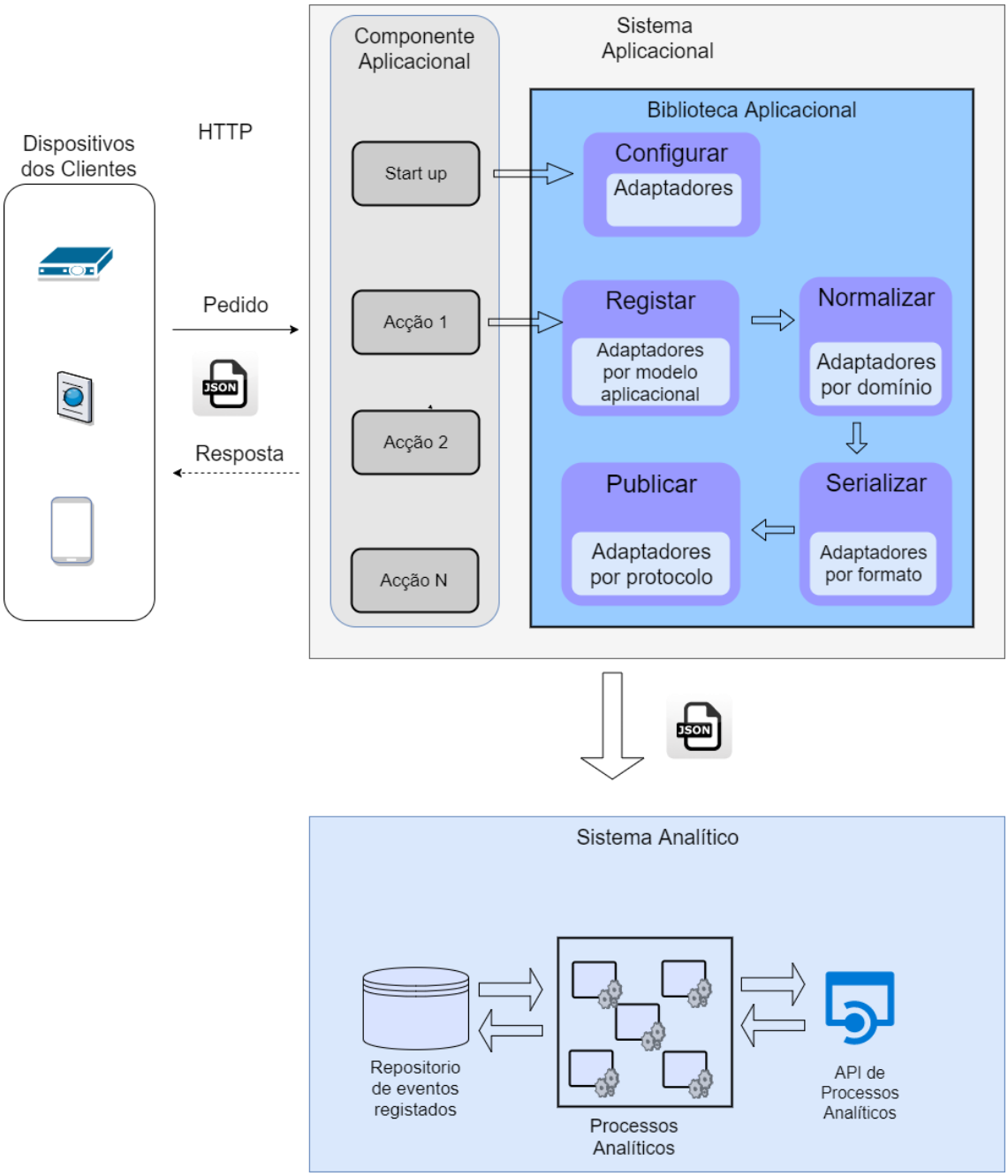

Figura 25 - Componentes da solução

Os componentes da biblioteca apresentados na Figura 25 é composta por cinco blocos que implementam as funcionalidades organizados em blocos funcionais da Figura 26:

1. Bloco 1 - Modelo físico criado com base na ontologia definida.
2. Bloco 2 – Vocabulário de entidades de domínio aplicacionais.

3. Bloco 3 – Infraestrutura de mapeamento do domínio aplicação para o vocabulário da ontologia.
4. Bloco 4 – API para registar associações das várias entidades do modelo físico do SSE.
5. Bloco 5 – Extensibilidade à infraestrutura ASP.NET Web API.

O modelo físico consiste no conjunto de entidades que persistem os dados registados e suportam o envio dos dados recolhidos para o servidor remoto de processamento analítico. Associado ao evento registado mantém-se também o domínio de informação das diferentes entidades, caracterizando dessa forma o seu domínio aplicacional com uma representação descritiva, designada por isso por metadefinição de domínio. Estes dois blocos devido ao seu âmbito transversal, recolha nos serviços aplicacionais e processamento analítico, é disponibilizado num único componente reutilizável comum a todo o âmbito da solução, designado por SSE.

A construção de um evento aplicacional acontece nos serviços aplicacionais, identificados na Figura 12. Na criação destas associações nos serviços aplicacionais é necessário mapear os domínios de informação dos valores extraídos para a sua representação descritiva disponível no modelo físico da biblioteca definida no componente SSE. Exprimir relações de conceitos na ocorrência de um evento e mapear conceitos de diferentes domínios são capacidades disponíveis apenas para serviços cliente, designando por isso ao componente onde essa funcionalidade é disponibilizada de SSE.Client.

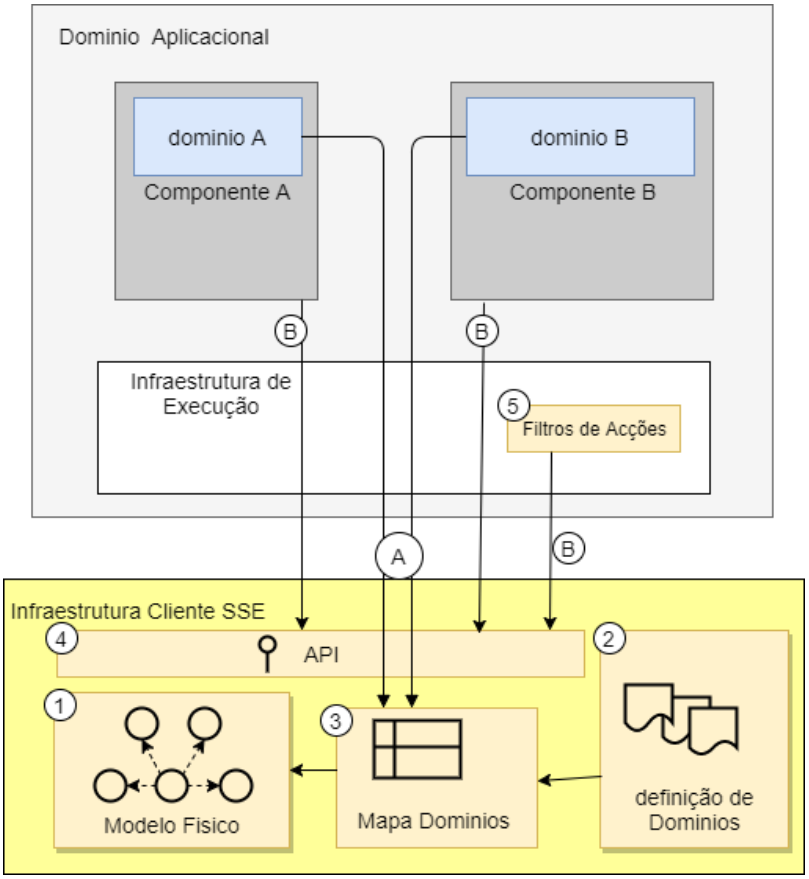

Figura 26 - Modelo conceptual da infraestrutura cliente SSE

A Figura 26 apresenta uma visão macro das funcionalidades e de como é que os componentes desta infraestrutura se relacionam entre si. A base desta infraestrutura é o modelo físico (1). Este modelo é a implementação da TBox da ontologia. É composta por um conjunto de classes compostas por propriedades que definem as relações identificadas na ontologia descrita na secção 4.7 e apresentada na integra no Anexo A. Este modelo físico é definido com as classes definidas na secção 4.7.2, representativo do conhecimento que se pretende registar. Foi identificado o requisito de poder mapear através de configuração por sistema, os conceitos dos dominios aplicacionais usado nos sistemas para o vocabulário da ontologia, para que todos os eventos recolhidos representem os conceitos com o mesmo vocabulário. Este requisito levou à implementação de uma infraestrutura de mapeamento, composta pela definição do vocabulário dos domínios (2) e por entidades com os quais se podem executar os mapeamentos (3) entre domínios. A descrição das classes que implementam este requisito é detalhado na secção 5.1.2.1. A utilização das entidades definidas nos pontos (1), (2) e (3) é abstraída por uma API (4) que garante que a criação das entidades que compõem o evento seja abstraído do programador. A utilização desta infraestrutura por parte das equipas de desenvolvimento fica então restrita à definição do mapeamento de quais os métodos da aplicação que serão mapeados em eventos (A), e à utilização da API (B) para registar informação do evento. Como já foi identificado na secção

4.2 um grande número dos sistemas do ecossistema consiste em serviços com pontos de comunicação em HTTP e como tal, para tirar partir da infraestrutura do ambiente de execução desses serviços esta biblioteca disponibiliza, uma implementação de um ponto de extensibilidade dessa infraestrutura, por forma a registar eventos de pedidos HTTP, sem alterar o código aplicacional existente, designados por filtros para intercepção da execução de acções da infraestrutura ASP.NET WebAPI [51, 52].

O filtro das acções, é a implementação de um ponto de extensibilidade da infraestrutura Web .NET. A necessidade deste componente está relacionado com o requisito de garantir que o processo de registo de eventos aplicacionais, impactasse o menos possível a implementação da lógica aplicacional. Com este componente, é possível implementar o padrão de separação de responsabilidades. Este ponto de extensibilidade permitem a execução de código customizado no encadeamento de operações que a infraestrutura executa para processar as mensagens HTTP de forma transversal.

Os blocos conceptuais descritos anteriormente estão presentes na biblioteca cliente SSE organizados em componentes físicos apresentados na Figura 27.

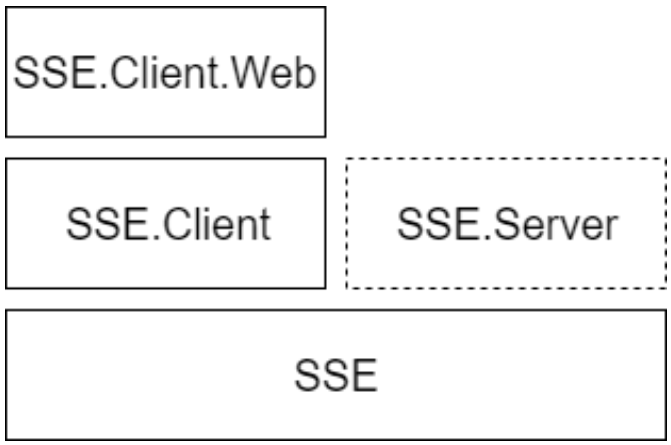

Figura 27 - Diagrama de módulos da infraestrutura cliente SSE

De seguida, apresentam-se em detalhe os módulos desenvolvidos Figura 27. Nesta descrição será enumerado quais os componentes existentes, qual a sua estrutura e quais a sua utilidade.

O objetivo deste conjunto de bibliotecas é disponibilizar às equipas de desenvolvimento de serviços implementados com a infraestrutura. NET um conjunto de funcionalidades que permitam a recolha, registo e envio para um sistema de processamento analítico de eventos aplicacionais relevantes para a análise do negócio da NOS. A distribuição destas bibliotecas será feita com base num repositório de bibliotecas reutilizáveis usado na NOS Inovação.

5.1.1 SSE

A biblioteca SSE contém a definição das entidades de domínio da infraestrutura, Este domínio é usado para persistir as características da ocorrência do evento aplicacional. Nesta biblioteca podemos encontrar a definição das classes usadas que compõem um agregado de objectos centralizado na entidade Event. As entidades existentes neste componente relacionam-se entre si tal como demonstrado pelo diagrama UML apresentado na Figura 28.

5.1.1.1 Entity

Entity é uma classe genérica [53, 54] que tem como objetivo guardar informação que descreve um conceito definido na ontologia. Esta entidade guarda informação identificadora do id único da entidade de domínio aplicacional. Desta forma consegue-se abstrair a representação de uma entidade do domínio aplicacional no domínio da ontologia da infraestrutura SSE.

Tabela 61 - Definição de Entity

Nome	**Tipo**	**Descrição**
Id	String	Valor que identifica a entidade no domínio aplicacional
Name	String	Nome do campo de onde foi lido o campo
Type	Generico T	Um dos enumerados definidos nesta biblioteca que descrevem um tipo possível de dados

5.1.1.2 Caller

Caller, é uma classe com a responsabilidade de guardar informação que caracteriza a aplicação cliente que originou a execução da acção. Esta entidade é uma representação física da classe *Caller* definida na ontologia no Capitulo 4.

Esta entidade é descrita pelos seguintes campos:

Tabela 62 - Modelo físico do contexto da aplicação cliente

Nome	**Tipo**	**Descrição**

Software	Software	Descrição da aplicação cliente
Device	Entity<DeviceType>	Descrição da identificação do device onde a aplicação cliente esta instalada
Network	Network	Descrição de informação de rede usada.
Session	Session	Id de sessão de login do utilizador
Language	String	Identificação da língua usada na aplicação

5.1.1.3 Callee

Esta entidade é uma representação física da classe *Callee* definida na ontologia no Capitulo 4. Caracteriza o contexto da ocorrência do evento no serviço aplicacional.

Tabela 63 - Modelo fisico da entidade Callee

Nome	Tipo	Descrição
Device	Entity<DeviceType>	Descrição da identificação do device
Software	Software	Descrição da indetificação do sistema onde ocorreu a evento
Component	Entity<ComponentType>	Descrição do componenente onde ocorreu o evento
Routine	Entity<RoutineType>	Descrição onde ocorreu a acção

5.1.1.4 OperationResult

Esta entidade é uma representação física da classe *Result* definida na ontologia no Capitulo 4. OperationResult, contem informação do resultado da execução da acção que originou o evento.

Tabela 64 - Modelo físico do resultado da operação do evento

Nome	Tipo	Descrição

Status	ResultType	Resultado da operação representado com status code
Duration	Timespan	Duração da execução da acção
Result	Entity<Domain>	Descrição da resposta devolvida pela acção
NumberOfResults	Numero de resultados	Numero de entidades retornadas

Nesta biblioteca são também definidas as entidades que tipificam os valores dos domínios de conhecimento apresentado na secção 4.6 relativos ao domínio de conhecimento de domínios aplicacionais, de verbos das acções do tipo de equipamentos, dos componentes e rotinas de código**Error! Reference source not found.** Essas entidades são classes que agrupam os valores possíveis de cada um dos tipos a descrever, apresentados no diagrama UML da Figura 29 e descritos na secção 4.6

Tabela 65 - Descritores de domínios

Nome	Descrição
Domains	Conceitos dos dominios aplicacionais da organização
ParticipantType	Tipo de Cliente envolvido no evento
DeviceType	Tipo de Devices
ComponentType	Tipo de componente onde o evento foi registado
RoutineType	Tipo de accção onde o evento foi registado
EventActions	Nomes do evento obtido da acção realizada

A Figura 28 apresenta o UML do modelo físico que se pretende representar baseado na ontologia definida. Ao contrário das linguagens de RDF ou OWL, UML é uma linguagem descritiva de

modelos mais visual com a qual nos permite descrever visualmente um modelo de um domínio aplicacional, sendo a linguagem utilizada na organização para esse efeito.

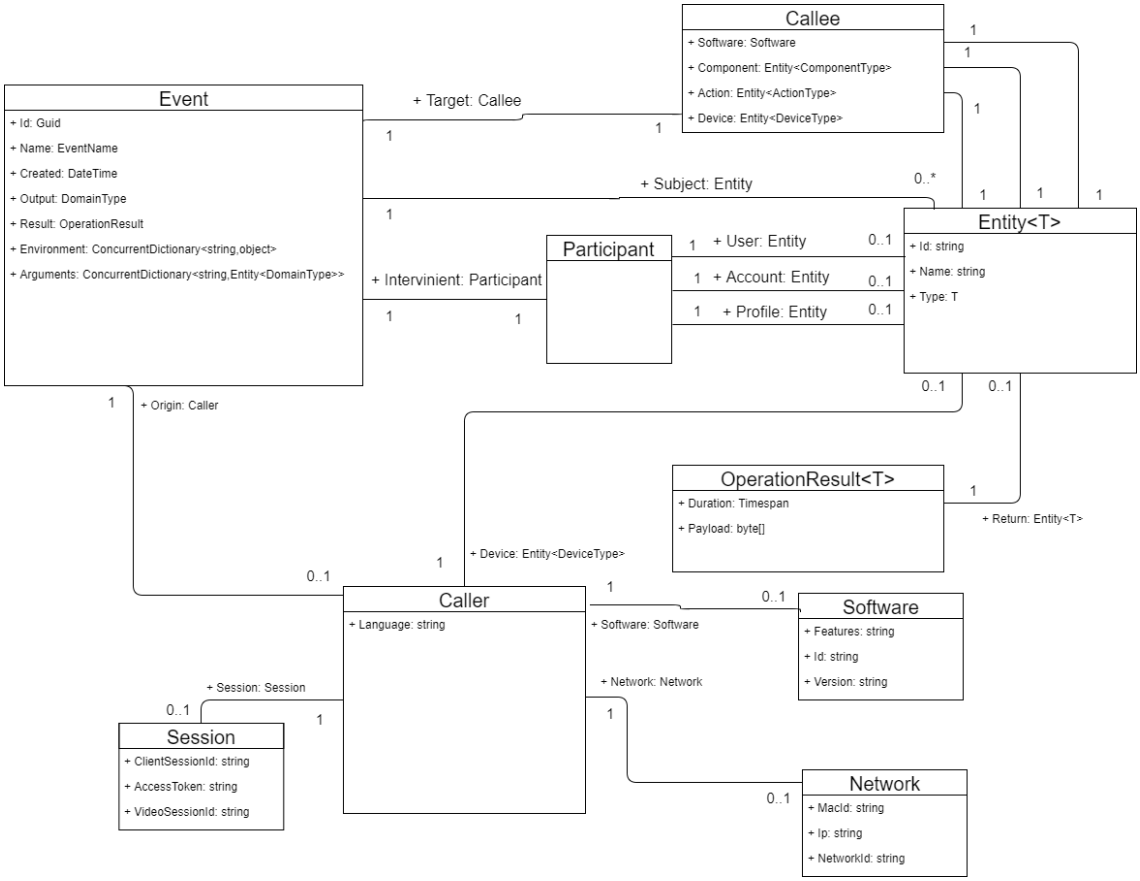

Figura 28 - Uml do módulo SSE

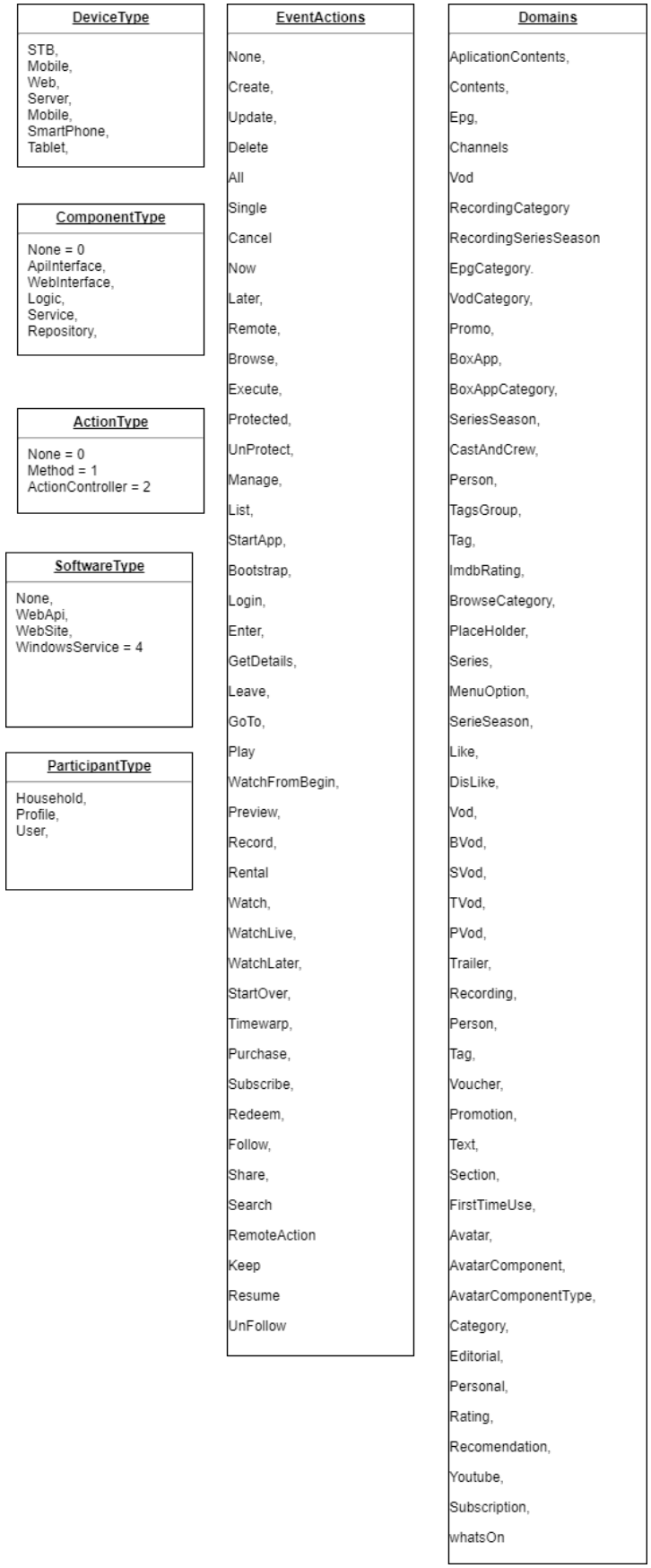

Figura 29 – UML de descritores de Tipos baseados no vocabulário da ontologia

5.1.2 SSE.Client

A biblioteca SSE.Client disponibiliza os componentes a serem usados numa aplicação cliente para a criação e registo dos eventos para processamento remoto. Estes componentes estão organizados em quatro espaços de nomes:

1. SSE.Client
2. SSE.Client.Mappings

O espaço de nomes SSE.client contem os componentes que serão as interfaces de utilização da equipa de desenvolvimento de um sistema na utilização desta biblioteca de código. Neste espaço de nomes, desenhado num padrão de Factory, existem as entidades desenhadas no UML da Figura 30.

A entidade *EventsLoggerProvider* implementa a interface *IEventsLoggerProvider e* ainda métodos que permitem registar quais os componentes que cada instância de *EventLogger* pedida vai usar.

Um *EventLogger*, quando é instanciado deve receber de *EventsLoggerProvider* as entidades:

1. Nome da classe onde o *EventLogger* foi instanciado através do *EventsLoggerProvider*
2. Identificação do *Device* dado pela instancia do tipo Entity<DeviceType>
3. Identificação do *Software* onde o EventLogger está a ser instanciado
4. Lista de todas as instâncias de *IEventMapMemberProvider*.
5. Lista de todas as instâncias de *IPublisher*

A criação de um evento acontece com a invocação do método *StartLogging* de *EventLogger* que recebe como parâmetro o contexto onde o evento está a acontecer. O contexto de execução para a criação de um evento deverá manter informação de:

1. Nome do método.
2. Dicionário de argumentos desse método.
3. Resultado da execução do método.
4. Instância do evento que está a ser criado.
5. Em caso de ter ocorrido uma excepção permite manter também informação dessa excepção.
6. Um booleano que indica se a operação esta a executar com sucesso.

Todos estes dados tem o objectivo de preencher dados do evento durante a execução dos métodos de *EventLogger*.

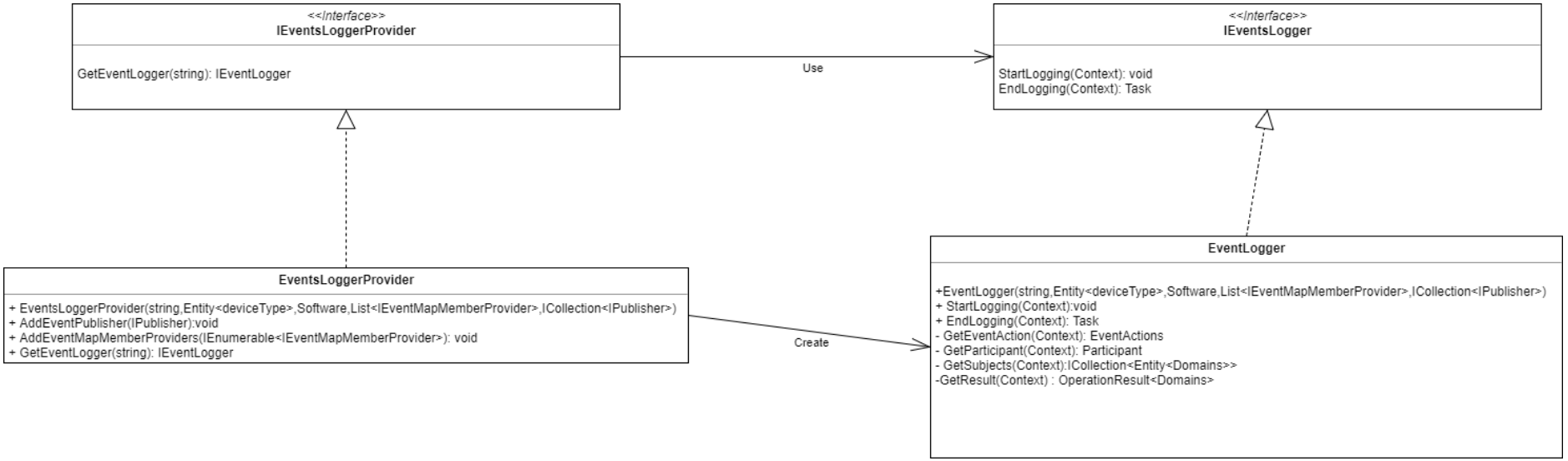

Figura 30 - UML EventLogger

5.1.2.1 SSE.Client.Mappings

Neste espaço de nomes encontram-se as entidades que permitem registar informação de um evento de forma customizada. O objectivo principal das classes deste espaço de nomes é o de mapear valores do domínio aplicacional para descrições desses valores que podem ser associados ao evento. As entidades deste espaço de nomes é listada na Tabela 66 - Entidades do espaço de nomes SSE.Client.Mappings que se relacionam como apresentado na Figura 31

Tabela 66 - Entidades do espaço de nomes SSE.Client.Mappings

Nome	**Tipo**	**Descrição**
IMapEventMember<out TOut>	Interface Genérica	Contrato da implementação do que deve ser uma entidade que executa o mapeamento.
IEventMapMemberProvider	Interface genérica	Contrato do registo e disponibilização de *IMapEventMember<out TOut>*
MapEventMembersProvider	Classe	Disponibiliza a configuração de uma lista de mapeamento de valores entre tipos
MapEventMember	Classe abstracta	Classe base que permite realizar o mapeamento entre o contexto da aplicação e o membro do evento específico.

A configuração dos mapeamentos deverá ocorrer antes da execução dos métodos de uma instância de *EventLogger*, de preferência na inicialização do processo em execução, garantindo-se dessa forma que estará disponível para todas as ocorrências possíveis.

O Mapeamento de membros dos eventos é realizado por instâncias de objectos que implementam a interface *IMapEventMember<object>*. Estas instâncias de objectos tem associado a elas um *scope* de utilização, que caso seja global deve executar o seu método *MapMember* independentemente da classe ou do método, caso contrário o *scope* é focado apenas para a classe e método que a instância do objecto indicar. Cada instância de objecto desta interface tem ainda uma propriedade que identifica o nome do Membro da Entidade Event que vai criar, com base no mapeamento dos dados presentes no contexto passado como parâmetro ao método da interface *IMapMember*.

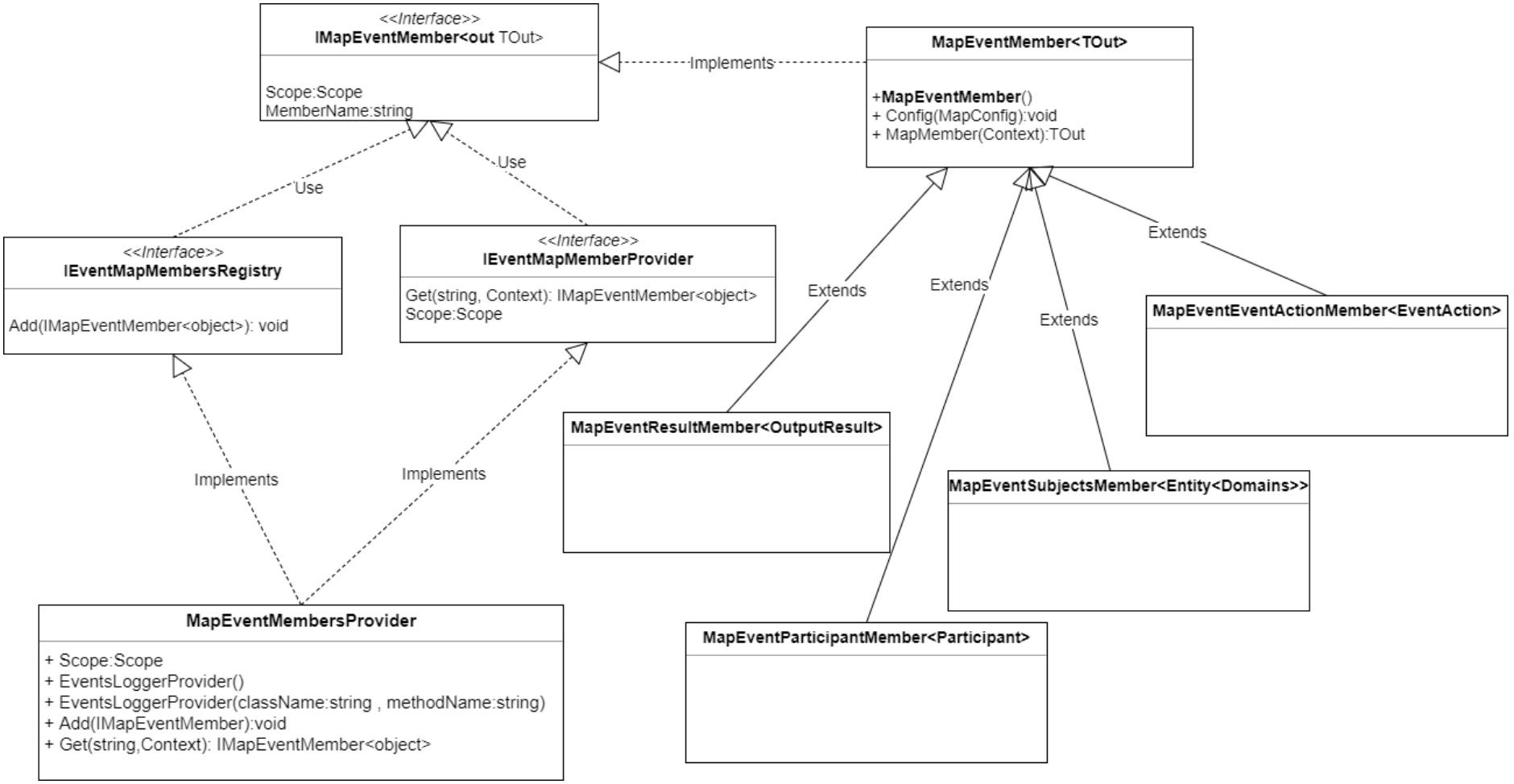

Figura 31- UML do espaço de nomes SSE.client.Mappings

Também neste caso a funcionalidade de disponibilizar mapeadores para os membros de um evento, foi disponibilizado no padrão factory apresentado no UML da Figura 31. A entidade *MapEventMembersProvider*, implementa o contrato que é usado durante o registo de todas as instâncias de objectos que implementam a interface *IMapEventMember<object>*, designada por *IEventMapMembersRegistry* e implementa ainda a interface *IEventMapMemberProvider* que permite obter uma instância de *IMapEventMember<object>* para executar o mapeamento de um valor para um membro de evento em especifico.

A utilização destas entidades acontece em dois momentos: (i) arranque da aplicação, (ii) processamento de um evento aplicacional.

No arranque da aplicação deverão ser registados os Mapas, *IMapEventMember<out TOut>*, e os mapeadores *IEventMapMemberProvider* que vão executar por *scope*.

Um mapa é usado para configurar o mapeamento de valores do serviço aplicacional com valores de descritores disponibilizados na DLL SSE listados na Figura 29, e funciona da seguinte forma:

1. No arranque da aplicação a configuração consiste em associar a uma instância de um tipo que implementa a interface *IEventMapMemberProvider*. Existe uma classe para cada membro de *Event*.
2. Durante o processamento de um evento, a criação das instâncias de objectos de cada um dos membros de Event consiste em usar as instâncias de objectos *IEventMapMemberProvider* que façam correspondência pelo scope, e por cada membro de *Event* usa a implementação de *IMapEventMember* específica.

5.1.2.2 SSE.Client.Publish

Este espaço de nomes contém a definição de tipos que permitem a publicação dos eventos para o servidor.

Tabela 67 - Interfaces do espaço de nomes SSE.Client.Publish

Nome	Tipo	Descrição
IEventsPublisher	Interface	Definição do contrato de um publicador de eventos

5.1.2.3 SSE.Client.Publish.Http

Este espaço de nomes contém todas as entidades necessárias para enviar eventos para um servidor HTTP.

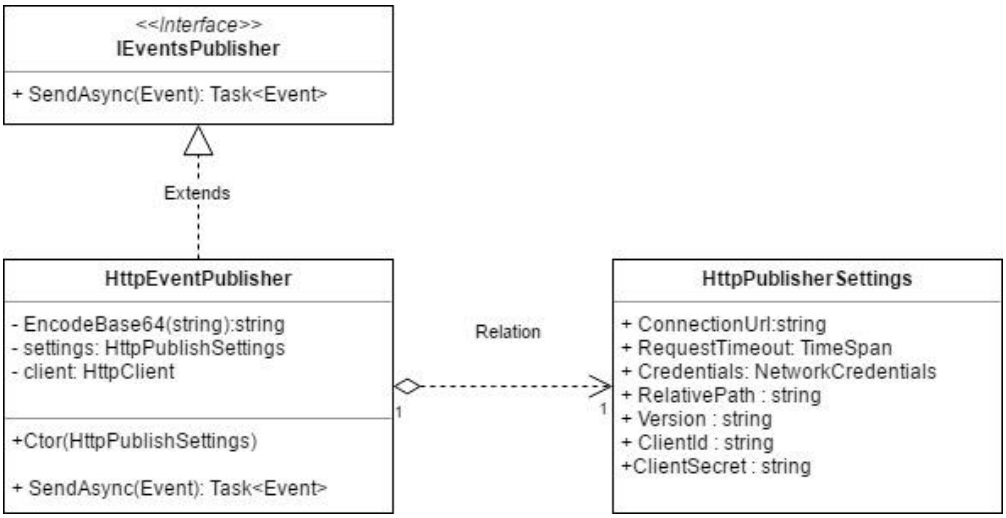

Figura 32 - UML do espaço de nomes SSE.Client.Publish.Http

Tabela 68 - Entidades do espaço de nomes *SSE.Client.Publish.Http*

Nome	**Tipo**	**Descrição**
HttpEventPublisher	Classe	Classe que envia eventos por http.
HttpPublishSettings	Classe	Classe que agrupa o conjunto de propriedades configuráveis para o envio dos eventos por HTTP

5.1.2.4 SSE.Client.Publish.Kafka

Kafka[19] é um sistema que consiste no barramento de dados que permite a recepção de mensagens a uma velocidade elevada e disponibiliza-as para consumo por diferentes modos, *stream* ou publicador/subscritor. Para esta comunicação foram implementadas as entidades listadas na Tabela 69 que se relacionam como apresentado na Figura 33.

Tabela 69 - Entidades do espaço de nomes SSE.Client.Publish.Kafka

Nome	**Tipo**	**Descrição**
KafkaEventPublisher	Classe	Classe que envia eventos por kafka.
KafkaPublishSettings	Classe	Classe que agrupa o conjunto de propriedades configuráveis para o envio dos eventos por kafka

[19] https://kafka.apache.org/

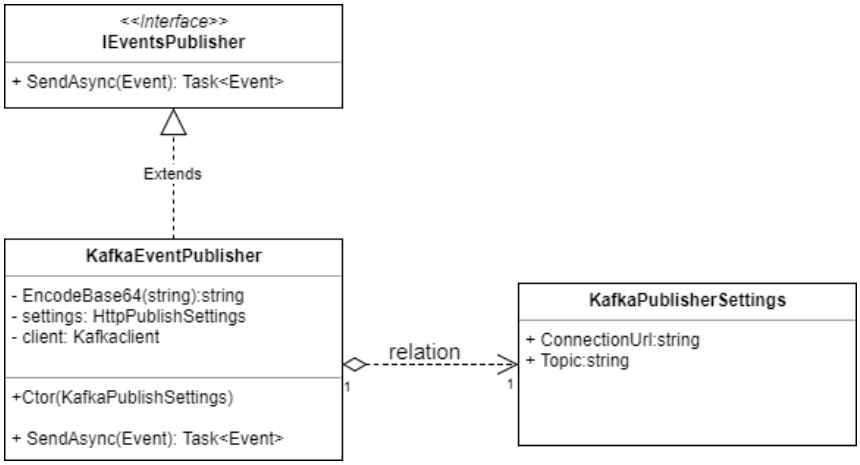

Figura 33 - Entidades do espaço de nos SSE.Client.Publish.Ka

5.1.3 SSE.Client.Web

A biblioteca SSE.client.Web disponibiliza componentes que são pontos de extensibilidade à infraestrutura ASP.NET Web API. Os pontos de extensibilidade usados são *ActionFilter*.

A pilha de componentes da infraestrutura ASP.NET Web API envolvida no processamento de um pedido HTTP consiste no que é apresentado na Figura 34, onde cada um desses componentes intervém pela ordem indicada pelos números.

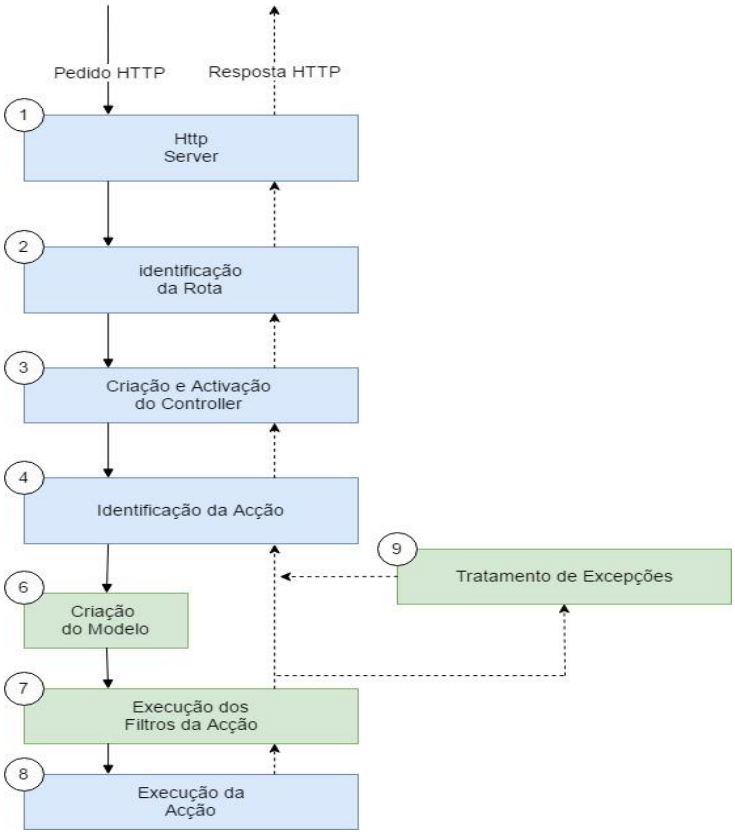

Figura 34 - Percurso de mensagem HTTP .NET WebApi

O sentido de fluxo de execução começa no HttpServer e vai até à *Action*, existindo a possibilidade de filtrar a execução desse fluxo a dois níveis:

1. Comum a todas as *actions* a serem executadas
2. Especificamente à *action* que se pretende.

A biblioteca SSE.Client.Web disponibiliza um componente que estende de *ActionFilter* apresentados na Figura 36. A entidade *StructuredSemanticEventFilter* tem como objectivo interceptar os dados de um pedido HTTP, sendo responsável por obter uma instância de *EventLogger* e usar essa instância para o registo do evento.

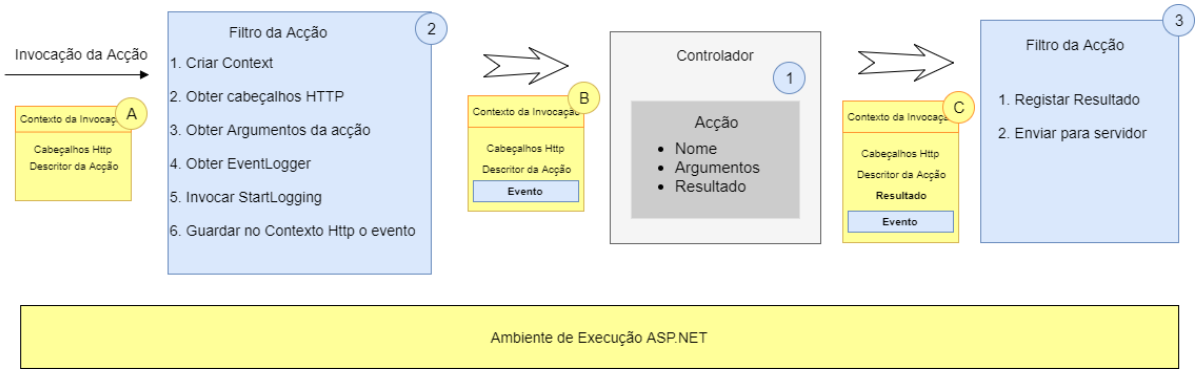

Figura 35 - Utilização do filtro da acção

Após a infraestrutura ASP.NET Web API ter instanciado o ApiController, identificado no pedido HTTP, e mapeado a acção que será executada, o fluxo da Figura 35 começa para o *ActionFilter* implementado nesta biblioteca (Figura 34 ponto 7). Nesta fase a infraestrutura ASP.NET Web API instancia todas as implementações da interface IActionFilter que foram registadas quando a aplicação arrancou e executa o método *ExecuteActionFilterAsync* que recebe como parâmetro uma instância de objecto da classe *HttpActionContext* (A). Este objecto, como o nome indica, contém o contexto do pedido HTTP que está a ser processado.

A utilização de filtros de acções minimiza o impacto em colocar código não aplicacional no fluxo aplicacional dos serviços e disponibiliza toda a informação do pedido HTTP relevante para o preenchimento de toda a informação que um Evento SSE necessita.

A execução do filtro da acção acontece em dois momentos:

1. Passo 2 - Antes do código aplicacional ser executado.
2. Passo 3 - Depois da execução do código aplicacional.

Antes da execução do código aplicacional, inicia-se o processo de registo de um evento. Primeiro obtém-se os dados para criar contexto da execução, nome da classe e método que vão ser executados. De seguida acrescenta-se a esse contexto, os argumentos da acção bem como os cabeçalhos HTTP. De seguida pede-se uma instância de EventLogger ao *IEventLoggerProvider* para o nome da classe, neste caso o nome do ApiController WebApi. Com a instância de *EventLogger* criada e obtida de *IEventLoggerProvider* é invocado o método *StartLogging* com o contexto previamente criado no *ActionFilter*. Antes de sair do âmbito de inicio da execução da

acção no Filtro da Acção, coloca-se o objecto do evento no contexto do pedido http, para transitar até ao momento em que a resposta é tratada. Após a execução do código aplicacional, acontece o segundo momento da execução do filtro. Neste ultimo passo (Passo 3) é recuperado do contexto da execução o evento guardado, e criado o resultado da operação no evento, bem como a descrição dos objectos do resultado da acção. No final do processo estar concluído, o evento é enviado para o servidor remoto configurado.

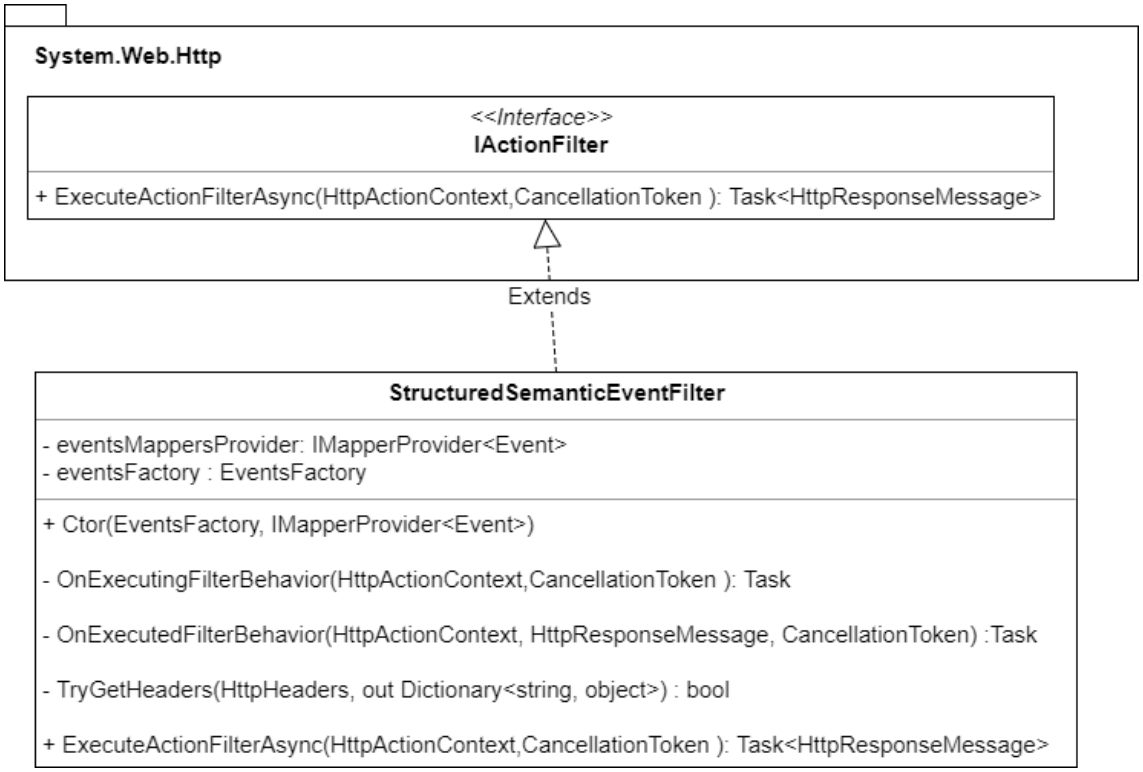

Figura 36 - UML SSE.Client.Web

A utilização deste filtro de acções é vantajoso porque remove do programador a responsabilidade de construir o contexto e gerir o tempo de vida da instância do objecto evento desde o inicio da execução da acção até ao fim.

Existe mais um espaço de nomes que é relevante para completar a utilização destes componentes - SSE.Serializers.

O Espaço de nomes SSE.Serializers, consiste num assembly que disponibiliza a capacidade de serializar o objecto em memória para o formato pretendido, sendo que o suportado actualmente

JSON. Este assembly contem os componentes apresentados na Figura 37 e utiliza a biblioteca Newtonsoft.Json para serializar os eventos registados para JSON.

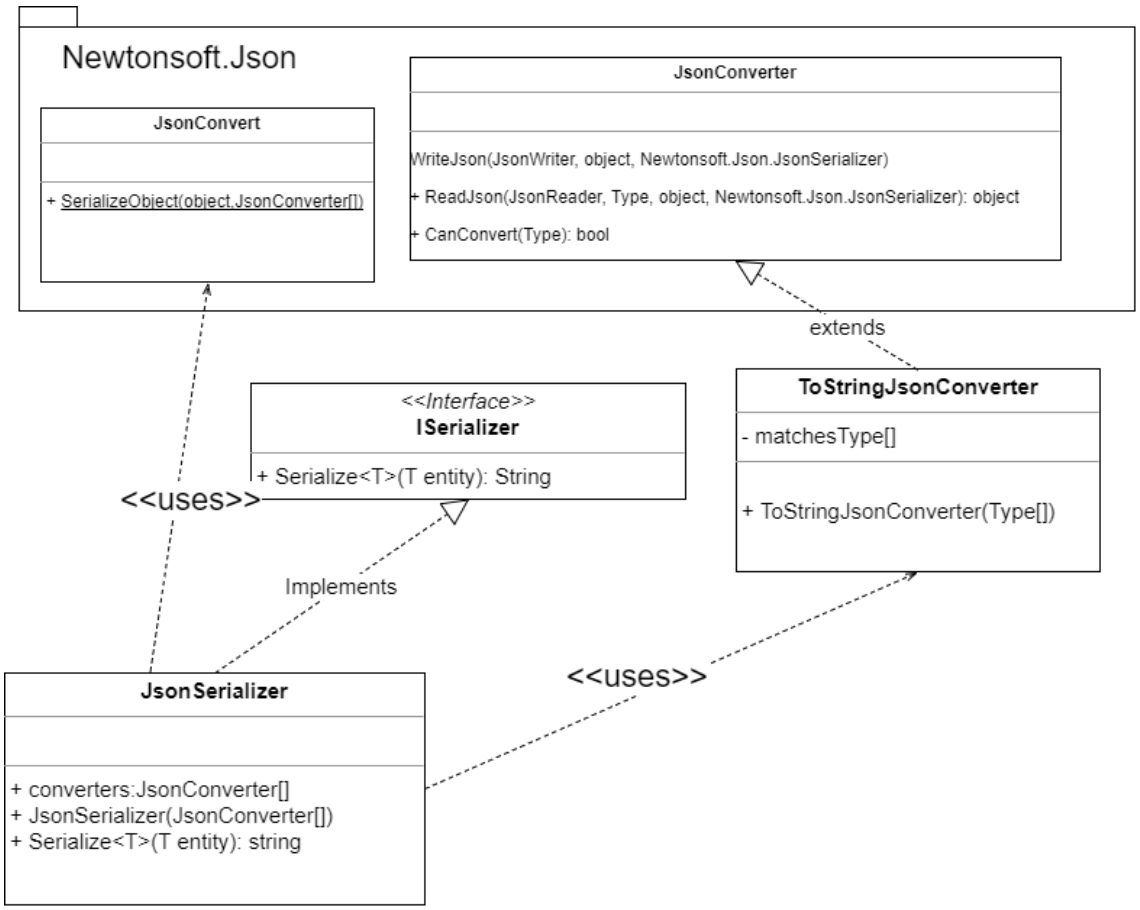

Figura 37 - SSE.Serializers UML

A entidade responsável por serializar o evento para JSON é a classe *JsonSerializer* que tem a capacidade de para meterizar a forma como o objecto é serializado com a utilização da implementação *ToStringJsonConverter* especifica de *JsonConverter*. O que o ToStringJsonConverter permite é alterar a forma como uma instância de um tipo é serializada. A forma como é alterada a serialização consiste em serializar a representação ToString de um objecto em vez de serializar todos os seus campos públicos. Este comportamento é importante para poder conseguir gerar um JSON de um evento sem informação desnecessária e que respeite o mais possivel a ontologia. Esta utilização é relevante para serializar o valor do campo Type das entidades Entity<T>, onde o T é ela uma classe com um campo privado value. A representação

de campos que sejam representado com esta entidade passa da representação da Listagem 47 para o que é representado na Listagem 48.

```
“Device”: {
    "Value": "00d0375d1031",
    "Type": { “value”:”STB” },
    "Name": "X-Core-DeviceId"
  },
```

Listagem 47 - *Device* serializado com *json* sem a utilização do *ToStringConverter*

```
“Device”: {
    "Value": "00d0375d1031",
    "Type": "STB",
    "Name": "X-Core-DeviceId"
  },
```

Listagem 48 - *Device* serializado para json com a utilização do ToStringConverter

A representação destes valores permite uma leitura dos campos directa, permitindo do lado do processo analítico operações sobre strings em vez de ter de manipular um objecto.

5.2 Resultados

A utilização desta biblioteca de código tem como objectivo representar ocorrências de eventos aplicacionais em JSON para que possam ser enviadas para servidores remotos big data para processamento analítico. Nesta secção será apresentada a utilização desta biblioteca, desde a instalação no projecto, a apresentação da configuração que a equipa de desenvolvimento terá de implementar e como devem criar um evento aplicacional para ser enviado para o canal de comunicação que seja configurado na aplicação. No final será comparado o resultado gerado por esta biblioteca e comparado com o que é usado actualmente, descrevendo as diferenças e vantagens da sua utilização.

A biblioteca é composta no seu total por nove módulos, todos eles após compilação com sucesso resultam num pacote nuget[20] que para ser utilizado deverá ser publicado para o repositório da

[20] https://www.nuget.org/

organização de pacotes nuget, quanto todos estes pacotes estão publicados a equipa de desenvolvimento tem acesso a estes pacotes através do IDE visual Studio pelo explorador de pacotes nuget, como na Figura 38.

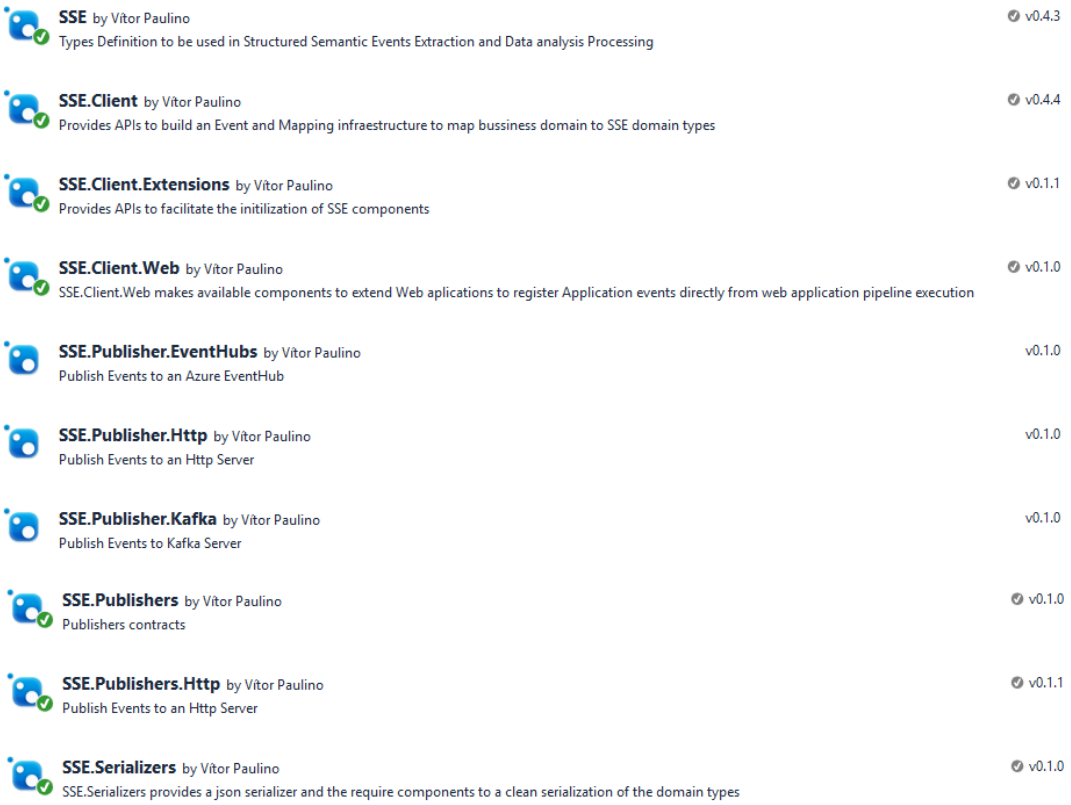

Figura 38 - Pacotes nuget da biblioteca de SSE

Para o funcionamento correcto das funcionalidades de registo de eventos aplicacionais, a equipa de desenvolvimento do projecto onde se estiver a adicionar estes pacotes terá de adicionar pelo menos o pacote SSE, e o *SSE.Client*. Para poder enviar o evento para um destino remoto, terão de ser adicionados os pacotes *SSE.Publishers* e *SSE.Serializer*, bem como o publisher que pretende usar, HTTP, Kafka ou *EventHubs*. O pacote do espaço de nomes *SSE.Serializer* será responsável por serializar para JSON o objecto do tipo SSE.Event e os componentes do espaço de nomes *SSE.Publisher* será responsável por enviar esse JSON para o destino que for configurado.

Após instalados com sucesso deverá ser feita a configuração dos componentes que vão ser usados pela aplicação. As configurações que são necessárias de se realizar são:

1. Configurar o mapeamento dos membros de um evento com origem no código aplicacional.
2. Configurar a instância do tipo que implementar *IPublisher* para publicar os eventos
3. Associar as instâncias dos objectos do ponto 1 e 2 à instância de *IEventLoggerProvider* que será disponibilizada na aplicação

4. Disponibilizar nos componentes da aplicação acesso a uma instância de IEventLoggerProvider.

5.2.1 Configurar mapeamento

A configuração do mapeamento de código aplicacional para membros do Tipo SSE.Event, acontece com recurso às entidades disponíveis no espaço de nomes SSE.Client.Mapping. A configuração deverá utilizar instâncias do tipo *SSE.Client.Mapping.MapEventMembersProvider* e podem ter dois níveis de alcance: (i) podem ser globais à aplicação (ii) ou apenas utilizáveis num método de uma classe.

Para que o mapeamento esteja disponível em qualquer método de qualquer instância de uma classe de uma aplicação, a criação destes componentes deverá ser realizada como apresentada na Listagem 49.

```
MapEventMembersProvider globalMembers = new MapEventMembersProvider();
```

Listagem 49 - Criar mapeamento global

Por cada membro da classe *SSE.Event* existe um método na classe *MapEventMembersProvider* que permite adicionar a configuração que permite indicar quais os argumentos do contexto da execução que deverão ser tidos em conta, e como é que eles podem ser convertidos para String.

```
 EventCallerMapConfig eventCallerMapConfig = new EventCallerMapConfig();
 eventCallerMapConfig.CallerDeviceConfig = new EventDeviceMapConfig()
{
    DeviceIdMemberName = "X-Core-DeviceId",
    DeviceTypeMemberName = "X-Core-DeviceType",
    DeviceIdValueGetter = (argument) => (argument as string[])[0].ToString(),
    DeviceTypeValueGetter = (argument) => (argument as string[])[0].ToString(),
};
 eventCallerMapConfig.CallerNetworkConfig = new EventNetworkMapConfig()
{

};

 eventCallerMapConfig.CallerSoftwareConfig = new EventSoftwareMapConfig()
{

};

eventCallerMapConfig.CallerSessionConfig = new EventSessionMapConfig()
{

};
globalMembers.AddCallerMapper(eventCallerMapConfig);
```

Listagem 50 - Configuração do mapeamento do membro Origin do tipo Caller do evento

Durante esta secção e a próxima os exemplos serão baseados nos campos apresentados na Listagem 7. Nesta listagem, todos os campos X-Core-*, são cabeçalhos HTTP e são transversais

à aplicação, ou seja, vão estar sempre presentes qualquer que seja o endereço HTTP da API REST da aplicação que seja invocado. Estes cabeçalhos enquadram-se em mapeamento global da aplicação, e como tal, serão mapeados usando a instância *globalMembers* da Listagem 49. O mapeamento dos cabeçalhos HTTP para os membros de um evento deverá acontecer como apresentado na Listagem 50. Para cada membro é criada uma instância de um tipo que estenda de

```
globalMembers.AddParticipantMapper(new EventParticipantMapConfig()
            {
                ParticipantType = ParticipantType.User,
                ParticipantIdMemberName = "X-Core-UserId",
                ParticipantIdValueGetter = (argument) => (argument as string[])[0].ToString(),

            })
            .AddParticipantMapper(new EventParticipantMapConfig()
            {
                ParticipantType = ParticipantType.Household,
                ParticipantIdMemberName = "X-Core-AccountId",
                ParticipantIdValueGetter = (argument) => (argument as string[])[0].ToString(),

            })
            .AddParticipantMapper(new EventParticipantMapConfig()
            {
                ParticipantType = ParticipantType.Profile,
                ParticipantIdMemberName = "X-Core-UserProfileId",
                ParticipantIdValueGetter = (argument) => (argument as string[])[0].ToString(),

              })
```

Listagem 51 - Configuração do mapeamento do membro Intervenient do tipo Participant do evento

MapConfig. Dos cabeçalhos HTTP, configura-se o *Origin* do evento, o *Intervenient*, todos os restantes são configurados específicos para por acção a registar. Para o caso da configuração do participante, os cabeçalhos HTTP disponibilizam várias entradas e todas elas podem estar preenchidas. Para mapear as várias possibilidades de configurar um participante, terá de ser adicionado cada uma das configurações do mapeamento por cada participante, como é apresentado na Listagem 51. A configuração de cada um dos mapeamentos dos membros consiste no nome do campo que existe nos argumentos do contexto, e numa expressão Lambda[21] que exprime de que forma é que o valor desse parâmetro deve ser obtido uma representação de string desse valor.

[21] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions

```
public class ChannelsController : ApiController

{
  [HttpPost]
  [Route("channels/{serviceId}/actions/{actionId}/submit")]
  public HttpResponseMessage ExecuteChannelAction(string serviceId, string actionId,
[FromBody] ActionParameter[] parameters)
        {
            ….
        }

}
```

Listagem 52 - Exemplo de uma acção aplicacional

Feita a configuração dos membros que são globais à aplicação é necessário configurar cada uma das acções que se pretende registar, como por exemplo a acção da Listagem 52. Para registar a configuração do mapeamento das acções deve-se instanciar um objecto do tipo *MapEventMembersProvider* mas utilizando o construtor que parametriza o nome da classe e do método para os quais corresponder a configuração que vai estar associada, como é o caso *ChannelsController*.

Dos membros da classe *SSE.Event* aqueles que são obtidos do contexto da execução, são:

1. Sujeitos da accção
2. Verbo da acção
3. Resultado da operação.

Considere-se o exemplo da Listagem 52. Na classe *ChannelsController* existe um método *ExecuteChannelAction*. Quando o URL deste método for invocado o atributo *IActionFilter* apresentado na secção 5.1.3, é executado. Caso exista uma instância de *MapEventMembersProvider* registado para a classe *ChannelsController* e para este método, *ExecuteChannelAction*, será criado um evento aplicacional que utilizará a configuração registada nessa instância e ainda a configuração global já apresentada na Listagem 50 e Listagem 51.

A configuração do nome do verbo da acção que é realizada neste método consiste em indicar qual o parâmetro de onde se lê o valor da acção. Este valor é depois mapeado para uma das acções conhecidas para que seja mapeado num dos valores pré-estabelecidos e definidos como valor do dominio de conhecimento dos verbos das acções na secção 4.6.10.

A configuração do sujeito também consiste no mesmo objectivo, mas neste caso identificar o nome do domínio aplicacional. Estes mapeamentos fazem parte da biblioteca.

No que diz respeito à configuração do resultado, a configuração consiste em indicar qual o domínio aplicacional que é retornado.

Para o método da Listagem 52 a configuração a registar seria a apresentada na Listagem 53.

```
MapEventMembersProvider executeChannelActions = new
MapEventMembersProvider("ChannelsController", "ExecuteChannelAction");

executeChannelActions.AddSubjectMapper(new EventSubjectMapConfig()
                {
                    Getter = (serviceIds) => (serviceIds as string),
                    InvocationArgumentName = "serviceId",
                })
                 .AddActionMapper(new EventActionMapConfig()
                 {
                   Getter=(actionId)=>actionId.ToString(),
                   InvocationArgumentName = "actionId"
                 })
                 .AddResultMapper(new EventResultMapConfig()
                {
                    ReturnHandler = (value) =>
                    {
                        if (value == null)
                            return null;

                        var entity = new Entity<Domains>() { Value = value.ToString(), Type
= Domains.None };
                        return entity;
                    }
                });
```

Listagem 53 - Configuração de um método de uma acção

Na configuração da acção de um evento ou do seu sujeito da acção, podem existir cenários onde a configuração do domínio aplicacional ou o nome da acção seja direto para o que o método faz. Nesses cenários a classe *EventActionMapConfig* e a classe *EventSubjectMapConfig* contém propriedades com as quais é possível mapear directamente um valor de *SSE.Types.EventActions* e *SSE.Types.Domains.*

Após realizada a configuração para os métodos dos quais se pretende recolher registo de terem ocorrido, é possível instanciar e executar o mecanismo de registo de eventos aplicacionais.

5.2.2 Configurar publicadores

A configuração do publicador que se pretende usar consiste em criar uma instância de uma das classes que implementa a interface *IPublisher*.

Caso se pretenda publicar eventos para um servidor HTTP, deve-se criar uma instância da classe *HttpEventPublisher* cujo único construtor disponível requer que seja passado como parâmetro um conjunto de configurações., nomeadamente: (i) Endereço base (ii) Endereço aplicacional (iii) Timeout do pedido, (iv) Credências, caso o servidor use autenticação (v) Versão da aplicação com a qual se pretende comunicar.

Para se publicar eventos para os EventHubs[22] do Azure[23] deve ser criar uma instância da classe *EventHubPublisher* que recebe como parâmetro o URL absoluto da instância do *EventHub* que está disponível no *Azure* e uma instância da classe *ISerializer* que vai ser usada para serializar a instância do evento recolhida e enviar na mensagem para o *EventHub*.

Por último existe o publicador para Kafka[24]. O kafka é um barramento de dados que suporta comunicação pelo padrão publicador subscritor. As aplicações clientes de um barramento de kafka publicam para um endereço absoluto e para um tópico que terá de ser previamente criado. Esta é uma tecnologia usada recentemente na organização que permite a recepção de dados a velocidades a um ritmo que se considera útil à natureza desta utilização.

Depois das configurações feitas, pode-se começar registar eventos.

5.2.3 Registar eventos aplicacionais

Para que uma aplicação realize o registo de eventos aplicacionais existe uma entidade do espaço de nomes que tem de ser criada e acessível a todos os componentes existentes numa aplicação. Todos os serviços desenvolvidos com a tecnologia .NET são implementados com o padrão de desenho inversão de controlo (IoC)[25]. Com este padrão de desenho consegue-se desacoplar os componentes através da definição de contratos que têm associados a si implementações concretas. IoC consiste em injectar nos componentes as suas dependências através de contratos, e em tempo de execução esses contratos são resolvidos para implementações concretas criadas em contentores que relacionado o contrato com a implementação a usar. A implementação desta biblioteca teve em conta este padrão de desenho.

O contrato que deve ser disponibilizado na aplicação é definido pela interface *SSE.Client.IEventsLoggerProvider* da Listagem 54. A implementação disponibilizada na biblioteca requere que a sua criação seja parameterizada com informação do tipo de Equipamento

```
public interface IEventsLoggerProvider
  {
      IEventLogger GetEventLogger(string className);
  }
```

Listagem 54 - Interface IEventsLoggerProvider

onde a aplicação está a correr, e também com informação da aplicação com instancias do tipo *SSE.Entity<DeviceType>* e *SSE.Software* respectivamente. Uma instância deste tipo tem como funcionalidades o de guardar as configurações apresentadas na secção 5.2.1 bem como os publicadores apresentados na 5.2.2. Este contrato permite aos componentes aplicacionais obterem uma instância de um outro componente que disponibiliza uma API para registar eventos

[22] https://azure.microsoft.com/en-us/services/event-hubs/
[23] https://azure.microsoft.com/pt-pt/
[24] https://kafka.apache.org/
[25] https://martinfowler.com/articles/injection.html

aplicacionais, que quando criado fica associado às instâncias dos objectos que implementam *SSE.Client.Mapping.IEventMapMemberProvider* respectivo, os globais e os específicos do nome da classe, bem como os publicadores configurados.

Uma instância de um objecto que implementa o contrato *SSE.Client.IEventLogger*, permite registar um evento através de dois métodos. StartingLogging e EndLogging.

```
public interface IEventLogger
{
    void StartLogging(Context context);
    Task EndLogging(Context ctx);
}
```

Listagem 55 - Código C# da interface IEventLogger

Estes dois métodos recebem como parâmetro o contexto onde o evento ocorreu apresentado na Listagem 56.

```
public class Context
{
        public string ClassName { get; private set; }
        public string MethodName { get; private set; }
        public object OperationOutput { get; set; }
        public Exception OperationException { get; set; }

        public IDictionary<string, object> Arguments { get; private set; }
        public Event Event { get; private set; }
        public bool? OperationSuccess { get; set; }
}
```

Listagem 56 - Contexto para o registo de eventos

Como já foi apresentado na secção 5.1.3, esta biblioteca permite a utilização de Filtros de acções que executam dois métodos quando uma acção de um pedido HTTP é realizado. Caso este filtro de acções seja utilizado, a equipa de desenvolvimento só necessita de realizar a configuração já apresentada e adicionar o filtro de acção à lista de filtros do ambiente de execução da aplicação. No que diz respeito à preparação do contexto e à sua manipulação durante a ocorrência do evento e a invocação dos métodos *StartLogging* e *EndLogging* são realizadas na implementação do Filtro *SSE.Client.Web.StruturedSemanticEventFilter*. Para uma visão global de todas as operações que estão envolvidas desde a configuração até à invocação do EndLogging esta disponível no Anexo B.

5.2.4 Resultado do evento gerado

Após o registo do evento que ocorreu na acção de um método aplicação, a biblioteca publica esse resultado usado as implementações de *SSE.Publishers.IPublisher*. O publisher utiliza uma implementação de *SSE.Serializers.ISerializer* que serializa o resultado para um formato de JSON. O resultado da serialização é então enviado para o canal de comunicação que a instância do tipo IPublisher definir, EventHubs, HTTP ou Kafka.

O resultado gerado consiste numa representação em JSON de uma instância de um objecto *SSE.Event*. Esta instância deste hoje representada em JSON caracteriza-se por ser a ABox da ontologia deste trabalho. Ocorrências de eventos aplicacionais serializadas para JSON são as instâncias da representação de conhecimento que definimos com a ontologia. Os campos que forem representados são aqueles que tinham instâncias de IEventMapMember associado, todos os outros ficam com os seus valores a NULL. Um exemplo de um registo pode-se encontrar no Anexo C. Este é uma representação de uma ocorrência de navegação no vídeoclube da *set-top box* UMA que não devolveu registos. Isto é possível identificar devido a uma estrutura com significado onde cada um dos campos tem um significado definido e onde a identificação do tipo dos valores aplicacionais está mapeado para um domínio bem definido. O *log* recolhido resolve os problemas identificados no *log* utilizado actualmente na organização. Todos os valores têm um significado associado, é uma representação modular onde a definição de tipos é reutilizável no próprio *log*, como é o exemplo da entidade *Software* e *Entity<T>*.

O resultado do *log* gerado para JSON tem uma estrutura sempre constante independentemente da aplicação, sendo o significado de cada um dos campos igual independentemente da aplicação. Campos simples, ou seja, representados por strings.

```
{
  "Id": "bc1dc514-617f-40f1-b1db-134d910f9339",
 "Action": "Browse",
 "Created": "2017-09-27T17: 03: 19.0709578Z",
 "Ended": "2017-09-27T17: 03: 30.0848595Z",
}
```

Listagem 57 - Campos simples do evento

Olhando para a definição do tipo que define a Action, este é uma entidade do tipo *SSE.Types.Domains* e como tal, a serialização JSON deste campo deveria ser um objecto complexo. Uma vez que a representação deste objecto é resultado o método ToString Implementou-se um newtonsoft Converter, SSE.Serializers.ToStringConverter que altera a serialização de tipos que forem indicados na construção to ToStringConverter para o seu método ToString, dessa forma, simplifica-se a representação em JSON dos valores. As datas por sua vez, estão identificadas como sendo UTC através da presença do valor Z na string. A representação

tipificada do zona da data vai permitir implementar correlações entre eventos baseados nas datas, sem ser necessário normalizar as datas à sua zona temporal.

A representação do interveniente na Listagem 58 apresenta cada um dos valores da entidade numa estrutura comum a todos eles.


```
"Intervenient": {
   "HouseHold": {
    "Value": "sa987654321",
    "Type": "HouseHold",
    "Name": "X-Core-AccountId"
   },
   "Profile": {
    "Value": null,
    "Type": "Profile",
    "Name": "X-Core-UserProfileId"
   },
   "User": {
    "Value": "123456789",
    "Type": "User",
    "Name": "X-Core-UserId"
   },
   "Other": null
  },
```


Listagem 58 - Representação JSON do campo Intervenient

Com uma representação homogénea a representar o mesmo tipo de informação, permite a criação de um processo analítico transversal a qualquer que seja o elemento que se está a tratar do membro *Intervenient*.

No que diz respeito aos restantes campos podemos constatar as mesmas observações. O campo *Origin*, *Target* e *OperationResult* são objecto complexos cuja sua estrutura é bem conhecida e será comum independentemente do evento que será capturado.

Esta abordagem para representar ocorrências de eventos aplicacionais revelou-se por ser uma abordagem vencedora. Permitiu resolver os problemas identificados no log Actual, suportados numa ontologia que por sua vez foram criados com base em conceitos já existentes de outras ontologias existentes que caracterizam eventos.

6 Conclusão

Este trabalho consistiu na especificação e no desenvolvimento de uma representação de *logs* aplicacionais baseados no conceito de evento. A especificação da representação de eventos aplicacionais foi baseada em lógica descritiva através da especificação da TBox e da ABox tal como apresentado na Figura 39. A TBox da base de conhecimento foi definida através de uma ontologia de domínio representada neste trabalho através de JSON-LD e a ABox é representada em JSON gerado através de uma biblioteca aplicacional desenvolvida para aplicações .NET multi-plataforma. Um evento, como vimos neste trabalho segundo vários autores, é uma entidade imutável que ocorreu ou esta prestes a ocorrer num local no qual participam um conjunto de entidades do meio onde esta inserido. Esta definição caracteriza um evento como sendo uma entidade rica em domínios de conhecimento tais como, temporal, local, objectos físicos. Esta riqueza de domínios de conhecimento fez com que a representação de um *log* semântico de um evento seja uma mais-valia para este trabalho, pois permitiu criar registos ricos em informação dos seus domínios associados. A TBox especificada neste trabalho é independente do sistema onde os eventos acontecem o que permite que seja usada em qualquer sistema da organização e até mesmo em sistemas de outras organizações. A extensibilidade da representação do *log* semântico também foi uma preocupação. A representação dos diferentes domínios de conhecimento em módulos vem permitir que esta ontologia seja extensível em cada uma das suas partes por quem a queira reutilizar. A representação simples de cada um dos domínios de conhecimento representados por três valores, nome, chave e valor, permite que outras organizações, ou pessoas possam estender para o seu domínio concreto de aplicação da ontologia, fazendo desta ontologia uma representação de conhecimento extensível.

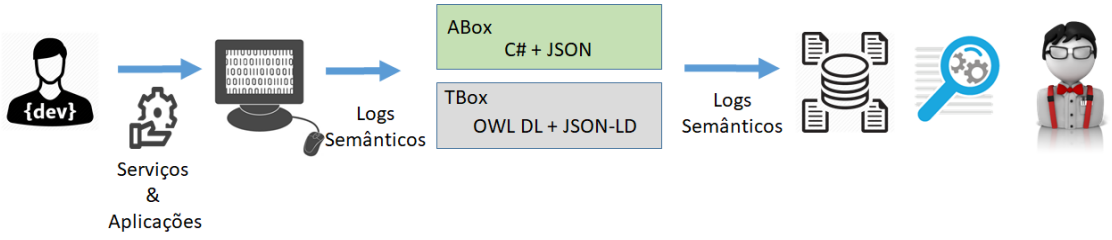

Figura 39 - Resumo da solução

Para a especificação da TBox, foi fundamental o suporte aplicacional existente para a recolha de *logs*. Para a criação uma nova representação e de uma plataforma homogénea para a recolha de *logs* das aplicações, era pretendido que houvesse retro compatibilidade com o que é recolhido das aplicações na actualidade. A análise do que é recolhido na actualidade e do que existe no processo analítico actual, foi importante para perceber qual o principal objectivo da organização na extração destes *logs*. Perceber as acções que o utilizador realiza enquanto interage com as aplicações é uma principais fontes de interesse da organização pois dessa forma consegue-se ter uma noção das tendências de consumo dos clientes. As limitações existentes neste processo estão

relacionadas com os diferentes formatos de *logs* que são extraídos das diferentes aplicações e também dos *logs* que o servidor IIS produz. A existência de duas fontes de dados por sistema e diferentes formatos de *logs* traz complexidade para o processo analítico e este também foi um problema abordado neste trabalho. Criar um log que seja composto por conceitos que abrangem toda a informação que é actualmente retirada dos *logs* do IIS e dos *logs* das aplicações irá simplificar o processo de normalização do processo analítico. O conteúdo desse *log* também foi alterado. Fez parte da definição da TBox a definição de um conjunto de conceitos que uniformiza a caracterização das entidades referidas nos *logs*, nomeadamente a tipificação dos nomes das acções, dos conceitos de domínios aplicacionais, do tipo de equipamentos que podem estar envolvidos, o tipo de componentes aplicacionais e a tipificação do resultado da acção que o evento caracteriza. A tipificação destes conceitos permite que a representação desses valores em qualquer aplicação ou sistema da organização seja homogéneo, sendo isso uma mais valia para o processo analítico. A normalização destes valores criou um vocabulário comum partilhado pelas equipas de desenvolvimento e pela equipa de analistas de dados que têm de recolher métricas baseadas em registos desta TBox. A importância de um vocabulário comum é importante especialmente quando serve de comunicação entre equipas distintas que partilham o mesmo domínio de negócio. A equipa de desenvolvimento de software garante que os *logs* gerados por essas aplicações, ABox, usam a TBox e as equipas de analise de dados podem construir processos CEP automatizados que gerem relatórios baseado nesse mesmo TBox.

A especificação da TBox foi feita com base em OWL DL. A utilização desta linguagem foi tida em conta porque permite a esta ontologia evoluir na definição da TBox dando uma maior expressividade do que se utilizasse RDFS, como por exemplo a utilização de *owl:sameAs* e *owl:AllValuesFrom* . Apesar de que fosse apenas necessário a utilização de RDFS para a definição da ontologia como ela é hoje, OWL é composta por um vocabulário mais extenso e rígido na definição de conceitos. Um exemplo disso é a diferença que existe em especificar uma entidade como sendo uma classe que deriva de outra versus uma entidade que é de um tipo exacto. Em RDFS não existe forma de distinguir porque ambas as opções são exprimidas da mesma forma com *rdfs:type*, enquanto em OWL se queremos exprimir que um individuo é de um tipo isso faz-se com recurso à definição de um elemento *owl:ObjectProperty*.

A definição da ontologia expressa em JSON-LD esteve relacionada com o facto da evolução de RDF 1.1 assim o sugerir e também por razões organizacionais. A representação em JSON é largamente adoptada em diferentes cenários, tais como, base de dados *NoSQL*, serialização de dados na comunicação entre sistemas e agora na modelação de dados. O formato JSON tem sido adoptado porque permite uma leitura fácil aos humanos e porque é uma representação que mapeia o paradigma orientado a objectos que é usado na maioria dos sistemas desenvolvidos na organização. A sintaxe de JSON-LD é baseada num conjunto de operadores e palavras-chave com um significado associado para que sejam interpretados por um processador de JSON-LD da forma

correcta e que permite ainda a extensabilidade a novos operadores consoante a utilização que se queira dar. No caso da nossa ontologia essa extensibilidade foi usada para a definição dos operadores da linguagem de OWL, como é o caso de owl:Cardinality, rdf#range ou rdf#domain. JSON-LD é uma representação já suportada pela ferramenta de referência para a construção de ontologias, Protégé, mas ainda pouco usada para representar ontologias. Apesar desta realidade considero ser um suporte válido pelas razões já aqui enumeradas.

A definição da ABox da nossa base de conhecimento está alinhado com um projecto interno na NOS Inovação. A necessidade de ter um conjunto de bibliotecas aplicacionais que sejam usadas para a geração de *logs* nos sistemas da organização. Este requisito foi atingido baseado na TBox definida neste trabalho. A infraestrutura desenvolvida é composta por um conjunto de bibliotecas capazes de executar em multiplataforma, ou seja, em aplicações .NET que estejam instaladas em sistemas operativos Windows e Linux. Os principais requisitos listados na secção 5.1 resultaram numa infraestrutura composta por um conjunto de módulos cada um com a sua responsabilidade coesa. O modelo físico implementado que mapeia a ontologia especificada é o centro desta infraestrutura. Quando se inicia a recolha de um log até ao seu fim são criadas instâncias desses objectos que mapeiam para as classes e propriedades definidas na ontologia. Para que ocorra a recolha desses valores o programador tem de escrever código bastante semelhante a duas bibliotecas que usa actualmente na organização, são eles o Log4Net e o AutoMapper[26]. A API que o programador usa com esta biblioteca para obter uma instância de um objecto para realizar o *log* de um evento assemelha-se à API que um programador usa com o Log4Net[27] para instanciar uma entidade de *ILogger*, Da mesma forma que o programador configura mapeamento entre entidades da aplicação quando utiliza a biblioteca AutoMapper, esta infraestrutura disponibiliza também uma API de mapeamento de informação da aplicação para entidades da ontologia. Com esta familiariedade pretende-se que o esforço de aprendizagem por parte de um programador da organização seja reduzido, podendo-se focar na tarefa principal que existe enquando utilizador desta infraestrutura que é a de configurar os acontecimentos e o que é que mapeia para a instância do objecto que representa o evento. Uma vez que todos os componentes que não são POCOs nesta biblioteca são implementações de contratos, isso possibilita a criação de entidades customizadas por aplicação que implementem esses contratos, fazendo desta infraestrutura extensível. Os principais pontos de extensibilidade que esta biblioteca disponibiliza são:

1. IMapEventMember
2. IPublisher
3. ISerializer

Com a evolução do modelo para casos concretos poderá ser necessário customizar a forma como se mapeia valores da aplicação para membros do evento, ou até mesmo com a evolução da

[26] http://automapper.org/
[27] https://logging.apache.org/log4net/

biblioteca e do modelo da ontologia seja necessário criar novas classes que estejam associadas a novas propriedades do evento, e para tal será necessário ter uma implementação específica de IMapEventMember para esse novo membro do evento.

O segundo ponto de extensibilidade que considero relevante é o IPublisher. Componentes que implementem este contrato são responsáveis por publicar o evento para um canal de comunicação. A NOS Inovação esta sempre a evoluir na forma como os sistemas comunicam entre si. Para garantir que esta infraestrutura acompanhe essa evolução é importante separar da componente do registo de eventos, da sua publicação para um local remoto. Caso surja um novo protocolo, o exista alguma especificidade na forma como se comunica com um servidor deve-se implementar uma classe que implemente esta interface, podendo dessa forma o evento ser enviado para um novo destino sem ser necessário alterar as restantes bibliotecas.

Por último mas não menos importante, temos o ponto de extensabilidade ISerializer. ISerializer veio permitir que a nossa solução não serialize apenas para JSON. Apesar de ser o formato actualmente disponibilizado, o facto de existir este ponto de extensibilidade permite que uma instância de objecto do tipo *Event* seja serializado para outro formato que seja necessário, como por exemplo para BSon, outro formato de JSON usado na organização, e ainda *protocol buffers* que começa agora a ser considerado em alguns casos de utilização dentro da organização.

No geral esta infraestrutura corresponde as nossas necessidades e permite-nos evoluir ao longo do tempo de forma modular e coesa, embora completamente *offline* da evolução da ontologia, ou seja, caso a ontologia evolua no futuro, desenvolvimentos paralelos têm de ser feitos sobre esta biblioteca. Esta característica apesar de ser uma desvantagem ao processo de automatização da evolução dos modelos, permite-nos manter *awareness* sobre a evolução da ontologia.

É objectivo da organização que mais bibliotecas cliente sejam criadas para outros sistemas ou aplicações que não sejam abrangidos pelo ambiente de execução .NET. A produção de *logs* baseados nesta ontologia nos ambientes de execução de dispositivos móveis, ou para ambientes de execução de *set-top boxes* são os próximos passos deste projecto na organização. Esta aposta por parte da NOS Inovação demonstra a importância e o impacto que esta ontologia vai ter na NOS Inovação no futuro próximo.

6.1 Trabalho futuro

Este trabalho teve como principal objetivo apresentar uma representação do domínio de conhecimento de eventos aplicacionais, ou seja, a definição de uma representação para ser usada em todas as aplicações apresentadas neste trabalho. A utilização da biblioteca de código permite que todas as aplicações, usando esta biblioteca, registem eventos aplicacional e os representem numa estrutura homogénea. Sendo homogénea, vai permitir que a execução de processos analíticos usando esta estrutura se torne mais simples, uma vez que remove a necessidade de existe processos de normalização de Logs específicos por cada aplicação emissora de um log,

existindo foco apenas no enriquecimento da informação recolhida. Um dos aspectos que fica para trabalho futuro, é a definição de um conjunto de axiomas e a implementação de um motor de lógica que saiba actuar sobre a ontologia deste trabalho e infira conhecimento dos vários exemplos que são recolhidos das aplicações e serviços. Para a implementação do motor de lógica, a ontologia apresentada poderia ser enriquecida com mais informação de características das propriedades e exercitar a sua utilização para perceber a sua mais valia, nomeadamente validar questões de performance, na inferência de relações entre entidades recolhidas de vários eventos recolhidos. Um dos exemplos da utilização seria regras de inferência a partir do *Participant* Inferir que dois utilizadores estão associadas à mesma *household*, permite extrair o conhecimento de que esses utilizadores no limite possam ser pessoas que habitam na mesma casa. Seria útil explorar também as relações que existam entre o sucesso ou insucesso de uma operação consoante o servidor onde os eventos são registados. A definição de um conjunto de regras de inferência orientadas à monitorização poderia ser vantajoso, sem recorrer a fontes de dados externas, ou seja, recorrendo apenas a registos recolhidos com este modelo, identificar quais os servidores que estão a causar erros, ou quais as aplicações que estão a causar erros consoante o participante, ou *software* usado pelo cliente.

As combinações possíveis de regras de inferência são tantas quantos os conceitos que definem esta ontologia, portanto considera-se que existe conhecimento que seja possível extrair com base em regras de lógica que sejam processadas sobre instâncias de eventos recolhidas desta ontologia.

Referências

[1] W. van der Aalst, "ProcessMining: Discovery,Conformance and Enhancement of Business Processes," em *Springer*, 2011.

[2] M. Fowler, Patterns of Enterprise Application Architecture, Pearson Education Inc, 2003.

[3] D. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise System, 2002.

[4] D. Gaaevic, D. Djuric e V. Devedzic, Model Driven Architecture and Ontology Development, 2006.

[5] T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," em *International Workshop on Formal Ontology,*, Italy, 1993.

[6] R. Suder, V. Benjamins e D. Fensel, "Knowledge Engineering:Principles and Methods," *Data & Knowledge Engineering,* vol. 25, pp. 161-197, 1998.

[7] B. C. Grau, B. Parsia, E. Sirin e A. Kalyanpur, "Modularity and web ontologies.," em *20th International Conference on Principles of Knowledge Representation and Reasonin*, 2006.

[8] D. Jones, T. Bench-Capon e P. Visser, "Methodologies for ontology development.," em *Proc.IT KNOWS Conference, XV IFIP World Computer Congress*, Budapest, 1998.

[9] T. Gruber, "A Translation Approach to Portable Ontology Specification," em *Knowledge Acquisition 55: 199-220.*, 1993.

[10] M. d'Aquin, A. Schlicht, H. Stuckenschmidt e M. Sabou, "Criteria and Evaluation for Ontology Modularization Techniques," em *Modular ontologies*, Berlin, Springer-Verlag, 2009., pp. 67-89.

[11] B. Chandrasekaran, J. R. Josephson e V. R. Benjamins, "What Are Ontologies, and Why Do We Need Them?," *IEEE Intelligent Systems,* vol. 14, pp. 20-26, 1999.

[12] M. Jarrar, "Towards Methodological Principles for Ontology Engineering.," Universidade de Bruxelas, 2005.

[13] A. Scherp, C. Saathoff, T. Franz e S. Staab, "Designing Core Ontologies," Germany, 2009.

[14] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator e W. Swartout, "Enabling Technology for knowledge sharing," *AI Magazine*, vol. 12, pp. 36-56, 1991.

[15] G. Vanheijst, A. Schreiber e B. Wielinga, "Using explicit ontologies in KBS development," em *Human – Computer Studies*, Amsterdam, 1997.

[16] N. Noy e D. McGuinness, “Ontology Development 101: A Guide to Creating Your ontology,” Stanford University, Stanford.

[17] Y. Raimond e S. Abdallah, “The Event Ontology,” 25 10 2007. [Online]. Available: http://motools.sourceforge.net/event/event.html. [Acedido em 22 05 2017].

[18] S. H. P. Stephen Cranefield, “UML-Based Ontology Modelling for Software Agents”.

[19] “OWL Guide,” [Online]. Available: https://www.w3.org/TR/owl-guide/.

[20] “Resource Description Framework (RDF),” [Online]. Available: https://www.w3.org/RDF/.

[21] “rdf syntax grammar,” [Online]. Available: https://www.w3.org/TR/rdf-syntax-grammar/.

[22] “JSON-LD 1.0 - A JSON-based Serialization for Linked Data,” [Online]. Available: https://www.w3.org/TR/json-ld/.

[23] “RDF AND JSON-LD UseCases,” [Online]. Available: https://www.w3.org/2013/dwbp/wiki/RDF_AND_JSON-LD_UseCases.

[24] W3C, “rdf-sparql-query,” [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/.

[25] I. Davis, T. Steiner e A. Hors, “RDF 1.1 JSON Alternate Serialization (RDF/JSON),” W3C Working Group, 11 2013. [Online]. Available: https://www.w3.org/TR/rdf-json/.

[26] “JSON-LD 1.1 A JSON-based Serialization for Linked Data,” July 2017. [Online]. Available: https://json-ld.org/spec/latest/json-ld/. [Acedido em 02 10 2017].

[27] M. Fox, “The TOVE Project: A Common-sense Model of the Enterprise,” em *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Berlin, Springer-Verlag, pp. 25-34.

[28] G.-P. A. J. N. Ferndndez. M, “METHONTOLOGY:From Ontological Art Towards Ontological Engineering,” AAAI Technical Report SS-97-06, 1997.

[29] M. C. Peraketh. B, “The IDEF5 Ontology Description Capture Method Overview,” *Knowledge Based Systems, Inc. (KBSI) Report, Texas,* 1994.

[30] A. GOMEZ-PEREZ, “A Framework to Verify Knowledge Sharing Technology,” *Expert Systems with Applications,* pp. 519-529, 1997.

[31] S. A. M. T. M. d. Sarra Ben Abbès, “Characterizing Modular Ontologies”.

[32] B. Grau, B. Parsia e E. Sirin, “Working with multiple ontologies on the semantic web,” em *International Semantic Web Conference,* 2004.

[33] C. Lagoze e J. Hunter, “The ABC Ontology and Model,” pp. 160-176, 2001.

[34] Y. Raimond e S. Abdallah, “The event ontology,” 2007. [Online]. Available: http://motools.sourceforge.net/event/event.html.

[35] “Ontology:DOLCE+DnS Ultralite,” 2010. [Online]. Available: http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite. [Acedido em 10 2017].

[36] A. Scherp, T. Franz, C. Saathoff e S. Staab, “F—A Model of Events based on the Foundational Ontology DOLCE+DnS Ultralite,” Alemanha.

[37] “EventsML-G2,” International Press Telecommunications Council., [Online]. Available: https://iptc.org/standards/eventsml-g2/.

[38] R. T. a. L. H. R. Shaw, “ Lode: Linking open descriptions of events,” em *Semantic Web, volume 5926,* 2009.

[39] “OpenCyc Brings Meaning to the Web,” Cycorp, [Online]. Available: http://www.cyc.com/about/media-coverage/opencyc-brings-meaning-web/.

[40] D. Brickley, “Basic Geo (WGS84 lat/long) Vocabulary,” [Online]. Available: https://www.w3.org/2003/01/geo/.

[41] J. R. H. a. F. Pan, “Time OWL Ontology,” 2011. [Online]. Available: http://www.w3.org/TR/owl-time/. [Acedido em 15 06 2017].

[42] S. M. A. T. H. S. Xiang-jun Wang, “Eventory -- An Event Based Media Repository”.

[43] “IPTC. EventML,” 2008. [Online]. Available: http://iptc.org/.

[44] A. Scherpa, S. Agaram e R. Jain, “Event-centric media management,” em *SPIE,* 2008.

[45] U. W. a. R. Jain., “Toward a common event model for multimedia applications,” em *IEEE MultiMedia,* 2007.

[46] K. T. a. A. Paschke, “Towards Semantic Event Processing,” em *Proceedings of the International RuleML Symposium on Rule Interchange and Applications,* 2009.

[47] S. G. G. D. A. A. D. A. K. ,. I. A. MARC SCHAAF, “Semantic Complex Event Processing,” ESTONIA.

[48] A. B. V. P. a. K. G. T. Zhu, “ Applying semantic web techniques to reservoir engineering: Challenges and experiences from event modeling,,” em *7th International Conference on Information Technology New Generations(ITNG),* 2010.

[49] M. L. N. a. N. P. Greis, “Rule-Based Complex Event Processing for Food Safety and Public Health,” em *RuleML,* 2011.

[50] J. L. a. X. Guan, “Complex event processing for sequence data and domain knowledge,” em *Proceedings of International Conference on Mechanic Automation and Control Engineering,* 2010.

[51] X. X. a. X. W. W. Si, “An ontology-based event matching dealing with semantic heterogeneity in pub/subsystems,,” em *Proceedings of the 4th International Conference on Computer Science Education (ICCSE)*, 2009.

[52] W. H. a. S. X. Z. Huaji, “Research on the ontology-based complex event processing engine of RFID technology for agricultural products,,” em *Proceedings of the International Conference on Artificial and Computational Intelligence (AICI), ,* 2009,.

[53] “Understanding Action Filters,” Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/understanding-action-filters-cs. [Acedido em 15 06 2017].

[54] B. W. S. A. ,. M. John Galloway, Professional ASP.NET MVC 5.

[55] A. Troelsen e P. Japikse, C# 6.0 and the .NET 4.6 Framework.

[56] B. Wagner e M. Wenzel, “Generics (C# Programming Guide),” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/. [Acedido em 06 2017].

[57] A. Gangemi, “DOLCE+DnS Ultralite,” [Online]. Available: http://www.ontologydesignpatterns.org/ont/dul/DUL.owl.

Anexo A

```
[ {
  "context"{
      "sse" : "http://www.semanticweb.org/vitorpaulino/ontologies/2017/6/sse",
     "owl" :"http://www.w3.org/2002/07/owl",
     "rdf" : "http://www.w3.org/2000/01/rdf-schema",
     "xmls" : "http://www.w3.org/2001/XMLSchema",
     "subClassOf": "owl:subClassOf",

  },
 "@id" : "_:genid1",
 "@type" : [ "owl#AllDisjointClasses" ],
 "owl#members" : [ {
   "@list" : [ {
```

```
      "@id" : "sse#Callee"
    }, {
      "@id" : "sse#Caller"
    }, {
      "@id" : "sse#ComponentType"
    }, {
      "@id" : "sse#DeviceType"
    }, {
      "@id" : "sse#Domain"
    }, {
      "@id" : "sse#Entity"
    }, {
      "@id" : "sse#Event"
    }, {
      "@id" : "sse#EventActions"
    }, {
      "@id" : "sse#Network"
    }, {
      "@id" : "sse#OperationResult"
    }, {
      "@id" : "sse#Participant"
    }, {
      "@id" : "sse#ParticipantType"
    }, {
      "@id" : "sse#RoutineType"
    }, {
      "@id" : "sse#Software"
    }, {
      "@id" : "sse#SoftwareType"
    } ]
  } ]
}, {
  "@id" : "sse",
  "@type" : [ "owl#Ontology" ],
  "rdf#comment" : [ {
    "@value" : "This ontology represents the domain of applications routine calls"
  } ]
}, {
  "@id" : "sse#ApiInterface",
  "@type" : [ "owl#NamedIndividual", "sse#ComponentType" ]
}, {
  "@id" : "sse#BrowseCategory",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#Browser",
  "@type" : [ "owl#NamedIndividual", "sse#ParticipantType" ]
}, {
  "@id" : "sse#Callee",
  "@type" : [ "owl#Class" ],
    "owl#subClassOf" : [ {
    "owl:Restriction" : {
          “owl:onProperty” : “sse:hasTarget”,
          “owl:Cardinality”: {
              “@id” : “1”,
              “rdf:datatype” : “xsd:nonNegativeInteger”
```

```
          }
    } } ],

}, {
  "@id" : "sse#Caller",
  "@type" : [ "owl#Class" ],
    "owl#subClassOf" : [ {
    "owl:Restriction" : {
          “owl:onProperty” : “sse:hasOrigin”,
          “owl:Cardinality”: {
              “@id” : “1”,
              “rdf:datatype” : “xsd:nonNegativeInteger”
           }
    } } ],
}, {
  "@id" : "sse#Class",
  "@type" : [ "owl#NamedIndividual", "sse#ComponentType" ]
}, {
  "@id" : "sse#ClientLanguage",
  "@type" : [ "owl#DatatypeProperty" ],
  "rdf#range" : [ {
    "@id" : "xmls#string"
  } ]
}, {
  "@id" : "sse#ComponentType",
  "@type" : [ "owl#Class" ]
}, {
  "@id" : "sse#Constructor",
  "@type" : [ "owl#NamedIndividual", "sse#RoutineType" ]
}, {
  "@id" : "sse#Content",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#Create",
  "@type" : [ "owl#NamedIndividual" ]
}, {
  "@id" : "sse#Created",
  "@type" : [ "owl#DatatypeProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Event"
  } ],
  "rdf#range" : [ {
    "@id" : "xmls#dateTime"
  } ]
}, {
  "@id" : "sse#Delegate",
  "@type" : [ "owl#NamedIndividual", "sse#RoutineType" ]
}, {
  "@id" : "sse#DeviceType",
  "@type" : [ "owl#Class" ]
}, {
  "@id" : "sse#Domain",
  "@type" : [ "owl#Class" ]
}, {
  "@id" : "sse#Dongle",
```

```
  "@type" : [ "owl#NamedIndividual", "sse#DeviceType" ]
},{
  "@id" : "sse#EId",
  "@type" : [ "owl#DatatypeProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Entity"
  } ],
  "rdf#range" : [ {
    "@id" : "xmls#string"
  } ]
},{
  "@id" : "sse#EName",
  "@type" : [ "owl#DatatypeProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Entity"
  } ],
  "rdf#range" : [ {
    "@id" : "xmls#string"
  } ]
},{
  "@id" : "sse#EType",
  "@type" : [ "owl#DatatypeProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Entity"
  } ],
  "rdf#range" : [ {
    "@id" : "xmls#string"
  } ]
},{
  "@id" : "sse#Ended",
  "@type" : [ "owl#DatatypeProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Event"
  } ],
  "rdf#range" : [ {
    "@id" : "xmls#dateTime"
  } ]
},{
  "@id" : "sse#Entity",
  "@type" : [ "owl#Class" ]
},{
  "@id" : "sse#Event",
  "@type" : [ "owl#Class" ]
},

{
            "@id" : "sse#Subject",
            "@type" : [ "owl#Class" ],
            "owl#subClassOf" : [ {
              "owl:Restriction" : {
        “owl:onProperty” : “sse:hasSubjects”,
        “owl:minCardinality”:
         {
            “@id” : “0”,
            “rdf:datatype” : “xsd:nonNegativeInteger”
```

```
          }
    } } ],
},
{
  "@id" : "sse#EventActions",
  "@type" : [ "owl#Class" ],
    "owl#subClassOf" : [ {
    "owl:Restriction" : {
          “owl:onProperty” : “sse:hasActionName”,
         “owl:minCardinality”:
          {
             “@id” : “1”,
             “rdf:datatype” : “xsd:nonNegativeInteger”
          }
    }
  } ],

}, {
  "@id" : "sse#HouseHold",
  "@type" : [ "owl#NamedIndividual", "sse#ParticipantType" ]
}, {
  "@id" : "sse#HttpEndpoint",
  "@type" : [ "owl#NamedIndividual", "sse#RoutineType" ]
}, {
  "@id" : "sse#Id",
  "@type" : [ "owl#DatatypeProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Event"
  } ],
  "rdf#range" : [ {
    "@id" : "xmls#base64Binary"
  } ]
}, {
  "@id" : "sse#Logic",
  "@type" : [ "owl#NamedIndividual", "sse#ComponentType" ]
}, {
  "@id" : "sse#MenuOption",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#Method",
  "@type" : [ "owl#NamedIndividual", "sse#RoutineType" ]
}, {
  "@id" : "sse#Mobile",
  "@type" : [ "owl#NamedIndividual", "sse#DeviceType" ]
}, {
  "@id" : "sse#Network",
  "@type" : [ "owl#Class" ]
},
{
             "@id" : "sse#hasSubjects",
             "@type" : [ "owl#ObjectProperty" ],
             "rdf#domain" : [ {
               "@id" : "sse#Event"
             } ],
             "rdf#range" : [ {
```

```
            "@id" : "sse#Subject"
          } ]
        }


,{
  "@id" : "sse#OperationResult",
  "@type" : [ "owl#Class" ],
   "owl#subClassOf" : [ {
   "owl:Restriction" : {
        “owl:onProperty” : “sse:hasResult”,
        “owl:maxCardinality”:
         {
            “@id” : “1”,
            “rdf:datatype” : “xsd:nonNegativeInteger”
         }
   } } ],

},
{
          "@id" : "sse#Parameter",
          "@type" : [ "owl#Class" ],
          "owl#subClassOf" : [ {
           "owl:Restriction" : {
        “owl:onProperty” : “sse:hasParameter”,
        “owl:minCardinality”:
         {
            “@id” : “0”,
            “rdf:datatype” : “xsd:nonNegativeInteger”
         }
   } } ],
}
, {
  "@id" : "sse#Participant",
  "@type" : [ "owl#Class" ],
   "owl#subClassOf" : [ {
   "owl:Restriction" : {
        “owl:onProperty” : “sse:hasParticipant”,
        “owl:Cardinality”: {
            “@id” : “1”,
            “rdf:datatype” : “xsd:nonNegativeInteger”
         }
   } } ],

}, {
  "@id" : "sse#ParticipantType",
  "@type" : [ "owl#Class" ]
}, {
  "@id" : "sse#Profile",
  "@type" : [ "owl#NamedIndividual", "sse#ParticipantType" ]
}, {
  "@id" : "sse#Rating",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#RecordingCategory",
```

```
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#RecordingSeriesSeason",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#Repository",
  "@type" : [ "owl#NamedIndividual", "sse#ComponentType" ]
 }, {
  "@id" : "sse#RoutineType",
  "@type" : [ "owl#Class" ]
 }, {
  "@id" : "sse#STB",
  "@type" : [ "owl#NamedIndividual", "sse#DeviceType" ]
 }, {
  "@id" : "sse#Server",
  "@type" : [ "owl#NamedIndividual", "sse#DeviceType" ]
 }, {
  "@id" : "sse#Service",
  "@type" : [ "owl#NamedIndividual", "sse#SoftwareType" ]
 }, {
  "@id" : "sse#Smartphone",
  "@type" : [ "owl#NamedIndividual", "sse#DeviceType" ]
 }, {
  "@id" : "sse#Software",
  "@type" : [ "owl#Class" ]
 }, {
  "@id" : "sse#SoftwareType",
  "@type" : [ "owl#Class" ]
 }, {
  "@id" : "sse#Tablet",
  "@type" : [ "owl#NamedIndividual", "sse#DeviceType" ]
 }, {
  "@id" : "sse#TagsGroup",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#User",
  "@type" : [ "owl#NamedIndividual", "sse#ParticipantType" ]
 }, {
  "@id" : "sse#Web",
  "@type" : [ "owl#NamedIndividual", "sse#DeviceType" ]
 }, {
  "@id" : "sse#WebApi",
  "@type" : [ "owl#NamedIndividual", "sse#SoftwareType" ]
 }, {
  "@id" : "sse#WebInterface",
  "@type" : [ "owl#NamedIndividual", "sse#ComponentType" ]
 }, {
  "@id" : "sse#WebSite",
  "@type" : [ "owl#NamedIndividual", "sse#SoftwareType" ]
 }, {
  "@id" : "sse#all",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#authorize",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
```

```
}, {
  "@id" : "sse#bootstrap",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#boxApp",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#boxAppCategory",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#browse",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#bvod",
  "@type" : [ "owl#NamedIndividual" ]
}, {
  "@id" : "sse#cancel",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#castAndCrew",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#channel",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#channelSettings",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#configuration",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#create",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#delete",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#device",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#entersection",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#epg",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#epgCategory",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#evaluate",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#execute",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
```

```
  "@id" : "sse#firstTimeUse",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#follow",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#get",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#getDetails",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#goTo",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#hasAction",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Event"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#EventActions"
  } ]
},
{
            "@id" : "sse#hasParticipant",
            "@type" : [ "owl#ObjectProperty" ],
            "rdf#domain" : [ {
              "@id" : "sse#Event"
            } ],
            "rdf#range" : [ {
              "@id" : "sse#Participant"
            } ]
          },

{
            "@id" : "sse#hasParameter",
            "@type" : [ "owl#ObjectProperty" ],
            "rdf#domain" : [ {
              "@id" : "sse#Event"
            } ],
            "rdf#range" : [ {
              "@id" : "sse#Parameter"
            } ]
          }
, {
  "@id" : "sse#hasTarget",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Event"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#Callee"
  } ]
}, {
```

```
  "@id" : "sse#hasCalleeComponent",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Callee"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#ComponentType"
  } ]
}, {
  "@id" : "sse#hasCalleeDevice",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Callee"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#DeviceType"
  } ]
}, {
  "@id" : "sse#hasCalleeRoutine",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Callee"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#RoutineType"
  } ]
}, {
  "@id" : "sse#hasCalleeSoftware",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Callee"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#SoftwareType"
  } ]
}, {
  "@id" : "sse#hasOrigin",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Event"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#Caller"
  } ]
}, {
  "@id" : "sse#hasCallerDevice",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Caller"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#DeviceType"
  } ]
}, {
  "@id" : "sse#hasCallerSoftware",
```

```
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Caller"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#SoftwareType"
  } ]
}, {
  "@id" : "sse#hasComponent",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Callee"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#Entity"
  } ]
}, {
  "@id" : "sse#hasHouseHold",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Participant"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#Entity"
  } ]
}, {
  "@id" : "sse#hasNetworkInfo",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#range" : [ {
    "@id" : "sse#Network"
  } ]
}, {
  "@id" : "sse#hasProfile",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Participant"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#Entity"
  } ]
}, {
  "@id" : "sse#hasResult",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Event"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#OperationResult"
  } ]
}, {
  "@id" : "sse#hasRoutine",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#range" : [ {
    "@id" : "sse#Entity"
  } ]
```

```
}, {
  "@id" : "sse#hasServer",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#range" : [ {
    "@id" : "sse#Entity"
  } ]
}, {
  "@id" : "sse#hasSoftwareFeatures",
  "@type" : [ "owl#ObjectProperty" ]
}, {
  "@id" : "sse#hasSoftwareId",
  "@type" : [ "owl#ObjectProperty" ]
}, {
  "@id" : "sse#hasSoftwareType",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Software"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#SoftwareType"
  } ]
}, {
  "@id" : "sse#hasSoftwareVersion",
  "@type" : [ "owl#ObjectProperty" ]
}, {
  "@id" : "sse#hasSubject",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Event"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#Domain"
  } ]
}, {
  "@id" : "sse#hasUser",
  "@type" : [ "owl#ObjectProperty" ],
  "rdf#domain" : [ {
    "@id" : "sse#Participant"
  } ],
  "rdf#range" : [ {
    "@id" : "sse#Entity"
  } ]
}, {
  "@id" : "sse#houseHold",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#image",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#insert",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#internet",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
```

```
  "@id" : "sse#keep",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#keepAll",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#like",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#list",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#login",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#logout",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#manage",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#many",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#none",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#person",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#personalRecording",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#placeholder",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#play",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#playFromBegin",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#playLater",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#playNow",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
 }, {
  "@id" : "sse#portfolio",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#portfolioProduct",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
 }, {
  "@id" : "sse#preview",
```

```
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#program",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#programEvent",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#promo",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#protect",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#purchase",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#pvod",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#query",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#quota",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#record",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#redeem",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#remove",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#rental",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#resume",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#rule",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#search",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#series",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#seriesSeason",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#share",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
```

```
}, {
  "@id" : "sse#single",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#startapp",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#subscribe",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#svod",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#tag",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#timewarp",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#unFollow",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#unProtect",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#update",
  "@type" : [ "owl#NamedIndividual", "sse#EventActions" ]
}, {
  "@id" : "sse#user",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#vod",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#vodCategory",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#voice",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#voiceline",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#webVideoData",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
}, {
  "@id" : "sse#youtubeVideo",
  "@type" : [ "owl#NamedIndividual", "sse#Domain" ]
} ]
```

Anexo B

```
// Inicio da configuração
List<IEventMapMemberProvider> mapMembersProviders = new List<IEventMapMemberProvider>();
// 1. Criar Fornecedor de mapeadores de membros do Evento especifico para a acção
BrowseChildren da classe // BrowseController
MapEventMembersProvider browserEventtBuilder = new
MapEventMembersProvider("BrowseController", "BrowseChildren");

            // 2. Adicionar configurações para os mapeadores
            // 2.1 Adicionar mapeador para sujeito da acção
            // 2.2 Adicionar mapeador da acção
            // 2.3 Adicionar mapeador do resultado da acção
            browserEventBuilder.AddSubjectMapper(new EventSubjectMapConfig()
                {
                    Getter = (nodeItemId) => nodeItemId.ToString(),
                    InvocationArgumentName = "nodeItemId",
                })
                .AddActionMapper(EventActions.Browse)
                .AddResultMapper(new EventResultMapConfig()
                {
                    ReturnHandler = (value) =>
                    {
                        return (value != null ? new Entity<Domains>() { Value =
value.ToString(), Type = Domains.Category, } : null);

                    }
                });
mapMembersProviders.Add(browserEventBuilder);
  // 3. Criar Fornecedor de mapeadores de membro de evento globais
            MapEventMembersProvider globalMembers = new MapEventMembersProvider();

            //4. Configurar mapeamento para o membro do evento que seja do Tipo Caller,
nomeadamente Origin
             // Este mapeamento é baseado em cabeçalhos HTTP que a aplicação recebe
            EventCallerMapConfig eventCallerMapConfig = new EventCallerMapConfig();
            eventCallerMapConfig.CallerDeviceConfig = new EventDeviceMapConfig()
            {
                DeviceIdMemberName = "X-Core-DeviceId",
                DeviceTypeMemberName = "X-Core-DeviceType",
                DeviceIdValueGetter = (argument) => (argument as string[])[0].ToString(),
                DeviceTypeValueGetter = (argument) => (argument as string[])[0].ToString(),
            };

            eventCallerMapConfig.CallerNetworkConfig = new EventNetworkMapConfig()
            {
                SourceIpArgumentName = "X-Core-ClientIp",
                SourceNetworkIdArgumentName = "X-Core-NetworkId",
                SourceNetworkIdArgumentGetter = (argument) => (argument as
string[])[0].ToString(),
                SourceIpArgumentGetter = (argument) => (argument as string[])[0].ToString(),
            };

            eventCallerMapConfig.CallerSoftwareConfig = new EventSoftwareMapConfig()
            {
                ApplicationIdArgumentName = "X-Core-AppId",
                ApplicationVersionArgumentName = "X-Core-AppVersion",
                ApplicationIdArgumentNameGetter = (argument) => (argument as
string[])[0].ToString(),
                ApplicationVersionArgumentNameGetter = (argument) => (argument as
string[])[0].ToString(),
            };

            eventCallerMapConfig.CallerSessionConfig = new EventSessionMapConfig()
            {
                  AccessTokenArgumentGetter = (argument)=> (argument as
string[])[0].ToString(),
                 VideoSessionIdGetter = (argument)=> (argument as string[])[0].ToString(),
                 ClientSessionIdGetter = (argument)=> (argument as string[])[0].ToString(),
                 AccessTokenArgumentName = "X-Core-Token"
```

```
globalMembers.AddParticipantMapper(new EventParticipantMapConfig()
            {
                ParticipantType = ParticipantType.User,
                ParticipantIdMemberName = "X-Core-UserId",
                ParticipantIdValueGetter = (argument) => (argument as
string[])[0].ToString(),

            })
            .AddParticipantMapper(new EventParticipantMapConfig()
            {
                ParticipantType = ParticipantType.Household,
                ParticipantIdMemberName = "X-Core-AccountId",
                ParticipantIdValueGetter = (argument) => (argument as
string[])[0].ToString(),

            })
            .AddParticipantMapper(new EventParticipantMapConfig()
            {
                 ParticipantType = ParticipantType.Profile,
                ParticipantIdMemberName = "X-Core-UserProfileId",
                ParticipantIdValueGetter = (argument) => (argument as
string[])[0].ToString(),

            }).AddCallerMapper(eventCallerMapConfig);

mapMembersProviders.Add(globalMembers);

             // Criação de uma entidade que sabe fornecer entidades de registo de eventos
var appVersion = Assembly.GetEntryAssembly().GetName().Version.ToString();
EventsLoggerProvider loggerProvider = new EventsLoggerProvider(
                                                    new Entity<DeviceTypes>()
                                                    {
                                                      Value = Environment.MachineName,
                                                      Name = "Environment.MachineName",
                                                      Type = DeviceTypes.Server
                                                    },
                                                    new Software()
                                                    {
                                                        Id =
Process.GetCurrentProcess().ProcessName,

                                                        Type = SoftwareType.WindowsService,
                                                        Features = "tests",
                                                        Version = appVersion
                                                    });
            // Configurar o fornecedor de registo de eventos com os mapeadores de membros de
evento
            loggerProvider.AddEventMapMemberProviders(mapMembersProviders);
             // Configurar o fornecedor de registo de eventos com o publicador

             JsonSerializerSettings jsonSettings = new JsonSerializerSettings();
            jsonSettings.Converters.Add(new ToStringJsonConverter(typeof(ComponentType),
typeof(DeviceTypes), typeof(Domains), typeof(EventActions), typeof(ParticipantType),
typeof(RoutineType), typeof(SoftwareType), typeof(ResultType)));
            loggerProvider.AddEventPublisher(new HttpEventPublisher(new
HttpPublishSettings()
            {
                 ConnectionUrl = "http://ct-cloud-be1/events",
                  RelativePath = "/Publish",
                   JsonSettings = jsonSettings,
            }));

// Fim da configuração

// Inicio da rotina de código que simula o conjunto de acções que teriam de se fazer,
nomeadamente recolha dos argumentos // da acção

var actionArguments = new Dictionary<string, object>();
            actionArguments.Add("nodeItemId", "browsecategory.personal@cipk123123123");
            actionArguments.Add("X-Core-UserId", new string[] { "123456789" });
            actionArguments.Add("X-Core-AccountId", new string[] { "sa987654321" });
            actionArguments.Add("X-Core-DeviceId", new string[] { "00d0375d1031" });
            actionArguments.Add("X-Core-DeviceType", new string[] { "stb" });
            actionArguments.Add("X-Core-AppId", new string[] { "NEXTGEN_E" });
            actionArguments.Add("X-Core-AppVersion", new string[] { "1.0.0-RC1b" });
            actionArguments.Add("X-Core-NetworkId", new string[] { "42833" });
            actionArguments.Add("X-Core-ClientIp", new string[] { "168.194.140.141" });
            actionArguments.Add("X-Core-UserType", new string[] { "profile" });
            actionArguments.Add("X-Core-Language", new string[] { "Port" });
            actionArguments.Add("X-Core-SessionId", new string[] { "qweqwe123123qwe" });
```

```
   // Obter uma instância de registador de eventos
            IEventLogger eventLogger = loggerProvider.GetEventLogger("BrowseController");

            // Iniciar
            eventLogger.StartLogging(ctx);

            // Terminar
            eventLogger.EndLogging(ctx);
```

Listagem 59 - Exemplo de operações para registar um evento

Anexo C

"{

 "Id": "bc1dc514-617f-40f1-b1db-134d910f9339",
 "Action": "Browse",
 "Created": "2017-09-27T17: 03: 19.0709578Z",
 "Ended": "2017-09-27T17: 03: 30.0848595Z",
 "Participant": {
  "HouseHold": {
   "Value": "sa987654321",
   "Type": "HouseHold",
   "Name": "X-Core-AccountId"
  },
  "Profile": {
   "Value": null,
   "Type": "Profile",
   "Name": "X-Core-UserProfileId"
  },
  "User": {
   "Value": "123456789",
   "Type": "User",
   "Name": "X-Core-UserId"
  },
  "Other": null
 },
 "Origin": {

```
  "Software": {
    "Id": "NEXTGEN_E",
    "Version": "1.0.0-RC1b",
    "Features": null,
    "Type": null
  },
  "Device": {
    "Value": "00d0375d1031",
    "Type": "STB",
    "Name": "X-Core-DeviceId"
  },
  "Network": {
    "Ip": "168.194.140.141",
    "NetworkId": "42833",
    "Url": null
  },
  "Session": {
    "ClientSessionId": "qweqwe123123qwe",
    "AccessToken": "12f3f3g3g4g45h5",
    "VideoSessionId": "123456789"
  },
  "Language": "Port"
},
"Target": {
  "Device": {
    "Value": "DESKTOP-EFVI9F4",
    "Type": "Server",
    "Name": "Environment.MachineName"
  },
  "Software": {
    "Id": "dotnet",
    "Version": "15.0.0.0",
    "Features": "tests",
    "Type": "WindowsService"
  },
  "Component": {
    "Value": "BrowseController",
```

```
    "Type": "Class",
    "Name": null
  },
  "Routine": {
    "Value": "BrowseChildren",
    "Type": "Method",
    "Name": null
  }
},
"Subjects": [
  {
    "Value": "vodcategory.personal@cipk123123123",
    "Type": "vod.Category",
    "Name": "nodeItemId"
  }
],
"Result": {
  "Status": "Success",
  "Return": [

  ],
  "NumberOfResults": 0,
  "Duration": "00: 00: 15.5221867"
},
"Environment": {

},
"Arguments": {
  "X-Core-DeviceId": [
    {
      "Value": "00d0375d1031",
      "Type": "String",
      "Name": "X-Core-DeviceId"
    }
  ],
  "X-Core-AccountId": [
    {
```

```
    "Value": "sa987654321",
    "Type": "String",
    "Name": "X-Core-AccountId"
   }
  ],
  "X-Core-UserType": [
   {
    "Value": "profile",
    "Type": "String",
    "Name": "X-Core-UserType"
   }
  ],
  "X-Core-SessionId": [
   {
    "Value": "qweqwe123123qwe",
    "Type": "String",
    "Name": "X-Core-SessionId"
   }
  ],
  "X-Core-Language": [
   {
    "Value": "Port",
    "Type": "String",
    "Name": "X-Core-Language"
   }
  ],
  "X-Core-AppId": [
   {
    "Value": "NEXTGEN_E",
    "Type": "String",
    "Name": "X-Core-AppId"
   }
  ],
  "X-Core-DeviceType": [
   {
    "Value": "stb",
    "Type": "String",
```

```
    "Name": "X-Core-DeviceType"
   }
  ],
  "X-Core-Lang": [
   {
    "Value": "Port",
    "Type": "String",
    "Name": "X-Core-Lang"
   }
  ],
  "X-Core-Token": [
   {
    "Value": "12f3f3g3g4g45h5",
    "Type": "String",
    "Name": "X-Core-Token"
   }
  ],
  "X-Core-UserId": [
   {
    "Value": "123456789",
    "Type": "String",
    "Name": "X-Core-UserId"
   }
  ],
  "nodeItemId": [
   {
    "Value": "vodcategory.personal@cipk123123123",
    "Type": "String",
    "Name": "nodeItemId"
   }
  ],
  "X-Core-NetworkId": [
   {
    "Value": "42833",
    "Type": "String",
    "Name": "X-Core-NetworkId"
   }
```

```
    ],
    "X-Core-VideoId": [
      {
        "Value": "123456789",
        "Type": "String",
        "Name": "X-Core-VideoId"
      }
    ],
    "X-Core-AppVersion": [
      {
        "Value": "1.0.0-RC1b",
        "Type": "String",
        "Name": "X-Core-AppVersion"
      }
    ],
    "X-Core-ClientIp": [
      {
        "Value": "168.194.140.141",
        "Type": "String",
        "Name": "X-Core-ClientIp"
      }
    ]
  }
}"
```