Um *Survey* sobre Extração de Esquemas de Documentos JSON

Rudimar Imhof[1], Angelo Augusto Frozza[1,2], Ronaldo dos Santos Mello[1]

[1]Departamento de Informática e Estatística (INE) – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.049-900 – Florianópolis – SC – Brasil

[2]Instituto Federal Catarinense (IFC) – Campus Camboriú
Caixa Postal 2016 – 88.340-055 – Camboriú – SC – Brasil

rudimar.imhof@gmail.com, frozza@ifc-camboriu.edu.br, r.mello@ufsc.br

Abstract. *JSON (JavaScript Object Notation) is a format for representation and interchange of complex and heterogeneous data that is becoming very popular. In this context, initiatives related to the integration or querying of large volumes of JSON data must deal with the problem of defining an unified schema for a set of relevant JSON documents. This paper aims to present a review on schema extraction from document in JSON format. The contributions of the paper are the presentation of approaches related to the theme as well as a comparative analysis of them.*

Resumo. *JSON (JavaScript Object Notation) é um formato de representação e intercâmbio de dados complexos e heterogêneos que vem crescendo em popularidade. Nesse sentido, iniciativas no sentido de integrar ou consultar grandes volumes de dados neste formato se deparam com a problemática de definir um esquema unificado para um conjunto de documentos JSON de interesse. Este artigo tem por objetivo apresentar um levantamento sobre extração de esquemas de documentos em formato JSON. As contribuições deste artigo são a apresentação de abordagens relacionados ao tema e uma análise comparativa destas abordagens.*

1. Introdução

Diversos autores têm manifestado que o movimento de bancos de dados voltados ao gerenciamento de dados complexos e semiestruturados tem crescido vertiginosamente. Como consequência deste crescimento, estima-se que boa parte dos dados, hoje no mundo, encontram-se armazenados em bancos de dados desse tipo [3]. Como exemplo, a utilização do Sistema Gerenciador de Banco de Dados (SGBD) *MongoDB*, um banco de dados (BD) NoSQL orientado a documentos, já superou a utilização do banco de dados relacional *PostgreSQL*, ficando atrás apenas de *Oracle*, *MySQL* e *Microsoft* SQL *Server* [1]. Entre as principais características desses bancos de dados estão a capacidade de representar dados complexos, a escalabilidade para gerenciar tanto grandes conjuntos de dados quanto o aumento do tráfego de dados, e a falta de esquemas ou o uso de esquemas flexíveis [12].

Nesse contexto encontra-se o formato de dados JSON (*JavaScript Object Notation*), que vem sendo utilizado cada vez mais para a representação e o intercâmbio de dados complexos e heterogêneos em diversos domínios de aplicação. Devido à

crescente utilização de bancos de dados NoSQL ou outros tipos de repositórios que suportam esse formato, surge a necessidade de integração ou acesso integrado a tais repositórios de dados.

Um problema associado a essas necessidades é a extração e unificação de esquemas de coleções de dados no formato JSON visando facilitar a futura manipulação desses dados. Assim sendo, este artigo apresenta um *survey* que descreve sucintamente e analisa trabalhos que visam a extração de esquemas de dados presentes para repositórios de dados no formato JSON. De forma complementar, é apresentado um quadro comparativo desses trabalhos e uma análise dos mesmos com base neste quadro. Além do problema de integração de dados já citado, esquemas extraídos de bancos de dados NoSQL podem ser úteis para a aquisição de conhecimento no desenvolvimento de sistemas, na reengenharia de sistemas, em processos de migração de sistemas e conversão de dados, entre outros [4].

Este artigo está organizado da seguinte forma: a seção 2 apresenta brevemente o formato de dados JSON. A seção 3 apresenta os trabalhos relacionados e suas principais características. A seção 4 apresenta uma análise comparativa dos trabalhos analisados. A seção 5 finaliza o estudo e apresenta as considerações finais.

2. O Formato de Dados JSON

JSON é um formato leve de intercâmbio de dados baseado na linguagem *JavaScript* [10]. Sua ascensão ocorreu devido à facilidade de leitura e escrita neste formato, tanto por humanos como por máquinas. O conteúdo de um documento JSON encontra-se em formato de texto, independente de linguagem de programação, mas que usa convenções que são familiares às usadas em linguagens como C, C++, *Java*, *Perl*, *Python*, entre outras. JSON é construído sobre duas estruturas compatíveis com estruturas existentes nessas linguagens de programação:

- Um conjunto de pares chave-valor, semelhante a um objeto ou registro;
- Uma lista ordenada de valores tratada como um *array*, vetor, lista ou sequência.

Figura 1 – Exemplo de documento no formato JSON

```
{
       "id": "00000234567894",
       "name": "Jane Doe",
       "birthday": "04/18/1978",
       "gender": "female",
       "type": "user",
       "work": [{
          "employer": {
             "id": "106119876543210",
             "name": "Doe Inc."
          },
          "start_date": "2007 - 08"
        },
        {
          "start_date": "2004",
          "end_date": "2007"
        }]
}
```

(Fonte: adaptado de [7])

A terminologia que define a estrutura de um documento JSON compreende os seguintes conceitos: (i) *Objeto*: um conjunto não ordenado de pares chave-valor. Um

objeto inicia com '{' e termina com '}', cada chave é seguida por ':' e os pares chave-valor são separados por ','; (ii) *Array*: uma coleção ordenada de valores que inicia com '[' e termina com ']', sendo cada elemento (valor) do *array* separado por ','; (iii) *Valor*: pode ser um tipo primitivo (*string*, número ou *booleano*), um valor nulo, um objeto ou um *array*. O uso de objetos e *arrays* como valores permite definir estruturas aninhadas.

A Figura 1 apresenta um exemplo de documento no formato JSON que descreve o perfil (*objeto*) de um usuário de rede social. As chaves *id*, *name*, *birthday*, *gender*, *type*, *start_date* e *end_date* possuem valores com tipos primitivos; o valor da chave *work* é um *array* com dois elementos do tipo *objeto*; o valor da chave *employer* também é um *objeto*.

3. Trabalhos Relacionados

Esta seção apresenta sucintamente, devido às limitações de espaço, as principais características dos trabalhos existentes na literatura que lidam com extração de esquemas de dados JSON. Para a seleção dos trabalhos, foi feita uma busca em sete bases de dados acadêmicas (*Science Direct*[1], ACM DL[2], IEEE *Xplore*[3], DBLP[4], Portal Periódicos CAPES[5], *Scopus*[6], *Web of Science*[7]) pelos termos "NoSQL", "*Reverse Engineering*", "JSON" e "NoSQL *Schema*". A busca limitou-se apenas a artigos em Inglês, publicados em Conferências e *Journals*, disponíveis publicamente. Após análise dos artigos, foram selecionados apenas os que apresentavam contribuições para o tema "engenharia reversa de bases de dados NoSQL".

O trabalho de Kapsammer *et al*. [7] tem por objetivo a extração e transformação de esquemas de perfis de usuário em redes sociais para a criação de perfis integrados. Para isso, propõe um processo em 4 etapas: (*i*) extração de instâncias de dados JSON - é obtido um conjunto de múltiplas amostras de dados diferentes (por meio de requisições às APIs – *Application Programming Interface* - de redes sociais), cada uma possuindo, eventualmente, fragmentos de dados complementares; (*ii*) extração de esquemas JSON das instâncias – múltiplos esquemas são extraídos dos registros obtidos na fase anterior. Esses esquemas são usados para conceber um único esquema consistente através de uma operação de *merge*; (*iii*) transformação dos esquemas JSON para esquemas no metamodelo ECORE [13] – um modelo canônico que possibilita a transformação de esquemas para outros formatos, como XML *Schema*, OWL (*Web Ontology Language*), de classe *Java* etc.; (*iv*) criação de perfis de usuário integrados – são aplicados processos de integração com o suporte de ferramentas de modelagem de propósito geral (como EA), ferramentas para verificação de similaridade (como COMA++) e para mapeamento de esquemas (como *MapForce*). Na etapa de extração de esquemas, o conhecimento prévio sobre o conjunto de esquemas pode ser utilizado para configurar estratégias de

[1] http://www.sciencedirect.com/
[2] http://dl.acm.org/ - opção "*The ACM Guide to Computing Literature*"
[3] http://ieeexplore.ieee.org/Xplore/home.jsp
[4] http://dblp.uni-trier.de/
[5] www.periodicos.capes.gov.br
[6] https://www.scopus.com/
[7] https://webofknowledge.com/

generalização e fusão. Esse conhecimento pode ser recuperado da documentação das APIs, das convenções de nomenclatura ou da investigação manual de exemplos.

O trabalho de Izquierdo & Carbot [5] apresenta uma abordagem para gerar o esquema básico de um conjunto de documentos JSON obtidos de APIs de serviços *Web*. O processo é composto por 3 etapas dirigidas por *Model Driven Engineering* (MDE): (*i*) pré-descoberta - extrai modelos JSON de baixo nível (usando *XText*[8], um *framework open source* para definir linguagens de programação, para definir um metamodelo JSON); (*ii*) descoberta de serviço único - visa a obtenção de informações de esquema para um dado serviço; (*iii*) descoberta multiserviço - encarregado de compor as informações de esquema obtidas na fase anterior, a fim de obter uma visão geral do domínio da aplicação. Na etapa *(i)*, documentos JSON são obtidos a partir de múltiplas chamadas a serviços de uma API e são transformados para um metamodelo JSON. Na etapa *(ii)*, o processo é executado para cada objeto JSON em dois modos de execução: criação de um novo esquema ou refinamento de um esquema existente. Seu produto é um modelo de domínio para cada serviço. Na etapa *(iii)* é realizado o *merge* dos modelos de domínio de serviço obtidos na etapa *(ii)*, produzindo o modelo de domínio da aplicação.

O trabalho de Kiran & Vijayakumar [8] propõe um sistema de integração semântico baseado em ontologia para bancos de dados NoSQL orientados a colunas, como o *HBase*. A abordagem de integração se dá por meio de um *endpoint* RDF sobre o qual se pode formular consultas usando ontologias combinadas obtidas de bancos de dados distintos. Apesar de seguir estratégias maduras para integração de bancos de dados relacionais, os autores destacam que há diferenças de implementação quando se trabalha com bancos de dados NoSQL. Considerando apenas o módulo de extração de esquemas, o trabalho propõe duas alternativas: a) geração de esquemas *online*, para bancos de dados sendo populados; b) geração de esquemas *offline*, para bancos de dados já populados. A extração de esquemas em cada banco de dados considera as seguintes atividades: (*i*) através de um *job MapReduce*, criar uma tabela de consulta no *HBase* para cada tabela que terá o esquema extraído; (*ii*) extrair o esquema com o suporte de um algoritmo genético que busca o esquema mais apto (a aptidão é definida como o número total de colunas do indivíduo e que possivelmente representa os demais indivíduos do banco de dados); (*iii*) criar um *HashMap in-memory* com os detalhes da família de coluna do esquema mais apto. O esquema obtido é então mapeado para uma ontologia OWL (esquema local). Uma vez que a proposta é de um sistema de integração de bancos de dados, em qualquer momento no sistema, dois esquemas existem: *a*) um esquema local para cada cliente; *b*) um esquema global acessível a todos os clientes.

O trabalho de Klettke *et al*. [9] propõe um algoritmo para extração de esquemas e produção de medidas de similaridade que capturam o grau de heterogeneidade de um conjunto de documentos JSON, além de revelar discrepâncias estruturais nos dados. O algoritmo é executado em 4 etapas: (*i*) seleção de documentos - executa a extração de esquemas sobre a coleção completa ou sobre um subconjunto de documentos JSON selecionados; (*ii*) extração da estrutura dos documentos JSON; (*iii*) construção do *Structure Identification Graph* (SG) – para cada conceito do documento JSON um nodo na árvore SG é adicionado ou estendido e uma aresta ligando ao nodo pai é adicionada ou estendida; (*iv*) criação do esquema JSON. Adicionalmente, os dados no grafo SG são

[8] http://www.eclipse.org/xtext

usados para produzir estatísticas, encontrar discrepâncias estruturais nos dados e calcular medidas que identificam quão regular são os documentos da coleção (grau de cobertura). O grau de cobertura é um valor entre 0 e 1 que especifica a sobreposição entre dois ou mais documentos, ou seja, quanto os documentos são similares. Discrepâncias estruturais permitem identificar estruturas que ocorrem raramente ou erros reais na estrutura de dados JSON.

O trabalho de Ruiz *et al*. [12] propõe um processo de engenharia reversa para bancos de dados NoSQL orientados a documentos através da metodologia MDE (*Model Driven Engineering*) [13]. O processo é realizado em 3 etapas: (*i*) extração de objetos JSON – através de um *job MapReduce* que extrai uma coleção de objetos JSON contendo um objeto para cada versão de uma entidade; (*ii*) definição do esquema JSON – cada objeto JSON é injetado em um modelo que está em conformidade com um metamodelo JSON proposto pelos autores. Isto é feito mapeando os elementos da gramática JSON em elementos do metamodelo; (*iii*) definição do esquema NoSQL – o processo de engenharia reversa é implementado como uma transformação *model-to-model* sobre cada modelo JSON, gerando modelos de domínio (com diferentes versões) em conformidade com um metamodelo de esquema NoSQL. Os modelos de esquema NoSQL inferidos podem ser usados para construir utilitários de bancos de dados, que exigem conhecimento da estrutura do BD, como, por exemplo, SQL *query engines*, e, ferramentas auxiliares, como validadores de dados, *scripts* de migração ou diagramas de esquemas.

O trabalho de Wang [14] apresenta um *framework* de gerenciamento de esquemas para bancos de dados NoSQL orientados a documentos. O *framework* descobre e persiste esquemas de documentos JSON e também suporta consultas sobre os esquemas e sumarização de esquemas. O trabalho propõe uma nova estrutura de dados, chamada *eSiBu-Tree*, para armazenar e dar suporte a consultas a esquemas, bem como o conceito de *skeleton* para visualização de esquemas, o qual representa o menor conjunto de atributos que melhor definem o núcleo do esquema. Documentos JSON (chamados registros JSON) são obtidos a partir de um *Dataset* JSON e, para cada registro, um *record schema* é extraído por meio da representação da estrutura do documento na *eSiBu-Tree*. Apenas os *labels* dos campos são considerados para montar a estrutura do registro na *eSiBu-Tree* (não fazendo referência a tipos de dados). O *skelenton* é usado para gerar um esquema único (*core schema*) do objeto, o qual é formado pelos atributos que aparecem com maior frequência nos esquemas dos registros.

O trabalho de Discala & Abadi [2] propõe um algoritmo que automaticamente transforma dados aninhados e desnormalizados, comumente encontrados em bancos de dados NoSQL, em dados relacionais que podem ser armazenados em um banco de dados convencionais. O algoritmo descobre dependências funcionais entre os atributos a fim de organizar esses atributos em tabelas relacionais. O processo apresenta 3 etapas: (*i*) criação de uma árvore de atributos e mineração de dependências funcionais entre atributos - as dependências são usadas para identificar grupos de atributos que podem corresponder a uma entidade independente que, posteriormente, dá origem a uma tabela; (*ii*) identificação de entidades de domínio sobrepostas – pesquisa a árvore de atributos para descobrir entidades semanticamente equivalentes espalhadas no conjunto de dados; (*iii*) agrupamento dos resultados intermediários para produzir o esquema físico relacional, mesmo que não esteja em conformidade com as tradicionais formas normais. Uma característica desse trabalho é que ele não faz uso da estrutura previamente existente em um documento JSON, buscando reconstruir a estrutura apenas pela identificação de

dependências funcionais presentes nos dados.

O trabalho de Liu *et al*. [11] apresenta o JSON *DataGuide*, que é um esquema auto computado, dinâmico e flexível, para coleções de documentos JSON, implementado no *Oracle 12cR2 release*. Uma coleção de documentos JSON é armazenada em uma coluna com a restrição "*IS JSON*". Para cada coluna JSON é também armazenado um JSON *DataGuide*, que é incrementalmente atualizado conforme novos documentos são inseridos. O *Oracle* não usa os esquemas gerados para o armazenamento de dados JSON, mas sim, para permitir consultas sobre uma visão relacional dos dados JSON usando SQL/JSON, uma linguagem de consulta baseada em caminhos (*paths*) JSON DOM. Com isso, introduz-se um paradigma *"escreva sem esquema, leia com esquema"*. O JSON *DataGuide* mantém uma derivação de todos os *paths* estruturais hierárquicos existentes em uma coleção JSON, os tipos de dados e as estatísticas dos valores escalares das folhas. Um JSON *DataGuide* para uma única instância de documento JSON é obtido pela extração do "esqueleto" dos nós *containers* (objetos e *arrays*) da árvore JSON DOM. Valores escalares nas folhas são trocados pelo tipo e comprimento dos dados. O JSON *DataGuide* para uma coleção de documentos é obtido pela combinação (*merge*) das instâncias *DataGuide* dos documentos da coleção, removendo os *paths* da árvore duplicados e que tem o mesmo tipo de nó de árvore. Caminhos com tipo de nó de árvore diferente são considerados diferentes. A informação de dados escalares folha é combinada, eliminando conflitos de tipos de dados por meio da definição de um tipo mais geral e usando o maior tamanho. Adicionalmente, o *DataGuide* armazena informações estatísticas dos caminhos, tais como frequência, valores mínimos e máximos e número de valores nulos. Por fim, o *Oracle* fornece um conjunto de procedimentos PL/SQL que permitem projetar visões relacionais e colunas virtuais dos dados JSON a partir do JSON *DataGuide*. Observa-se que o JSON *DataGuide* é aditivo, isto é, ele não remove *paths* quando documentos JSON são excluídos. Funções SQL para calcular o JSON *DataGuide* dinamicamente sobre os resultados de qualquer consulta SQL que retorne um conjunto de documentos JSON também são disponibilizadas.

4. Análise Comparativa

O Quadro 1 apresenta um comparativo das principais características observadas nos trabalhos apresentados na seção anterior. Além do identificador da referência do trabalho, as características consideradas são as seguintes: (*i*) *Origem* (fonte de origem usada para obter documentos JSON); (*ii*) *Objetivo* (perspectiva de uso dos esquemas extraídos); (*iii*) *Abordagem* (abordagem adotada para obter o esquema final); (*iv*) *Modelo Intermediário* (se o processo usa algum modelo de dados intermediário); (*v*) *Modelo de Saída* (como o esquema é disponibilizado ao final do processo); (*vi*) *Unificação* (se existe um processo de unificação de vários esquemas JSON visando obter um esquema único); e (*vii*) *Etapas do Processo* (um resumo das etapas do processo de extração adotado).

Com relação à *fonte de origem* dos documentos JSON, percebe-se que os trabalhos se concentram em três categorias: *a*) dados provenientes de APIs de serviços *Web*; *b*) *Datasets* de documentos JSON; *c*) Bancos de dados NoSQL. As duas últimas categorias podem ser consideradas como do mesmo tipo de fonte, uma vez que representam coleções de documentos JSON. Uma característica das APIs é que dois ou mais objetos no mesmo ou em diferentes documentos JSON gerados por uma chamada ao mesmo serviço não necessariamente têm a mesma estrutura exata, ou seja, é possível que alguns documentos possuam somente um subconjunto dos metadados por causa de parâmetros e filtros pas-

Quadro 1: Comparativo dos trabalhos analisados

Id	Origem	Objetivo	Abordagem	Modelo Interm.	Modelo de Saída	Unif.	Etapas do processo
[7]	APIs de redes sociais	Integrar perfis de usuário em redes sociais	Transformação de modelos	JSON *Schema*	Modelo de classes (ECORE)	SIM	a) Extração de dados b) Extração de esquemas c) Transformação d) Integração
[5]	APIs de serviços *web*	Criar visão de domínio para os serviços da API	Transformação de modelos	Meta-modelo JSON (ECORE)	Modelo de domínio da Aplicação (ECORE)	SIM	a) Pré-descoberta de documentos JSON b) Extração de esquema do serviço c) Criação do esquema de domínio
[8]	*HBase*	Integrar bancos de dados	Organização em ontologia	*HBase*	OWL	SIM	a) Criação de uma tabela de consulta sobre os documentos JSON com *MapReduce* b) Escolha do esquema mais apto via algoritmo genético c) Mapeamento do esquema para uma ontologia local OWL d) Combinação de ontologias para formar uma ontologia global (integração)
[9]	*Dataset* no *MongoDB*	Criar ferramentas para manipular e gerenciar esquemas	Organização hierárquica (grafo)	*Structure Identification Graph* (SG)	JSON *Schema*	SIM	a) Seleção de documentos JSON b) Extração estrutura dos documentos c) Construção do grafo SG d) Geração do esquema JSON
[12]	*MongoDB*, *CouchDB* e *HBase*	Criar utilitários para BD NoSQL	Transformação de modelos	Meta-modelo JSON	Meta-modelo de esquema NoSQL	NÃO	a) Extração de objetos JSON b) Transformação para esquemas JSON c) Transformação para esquema NoSQL
[14]	*Datasets* JSON	Consultar esquemas e integrar dados	Organização Hierárquica (árvore)	*eSiBu-Tree*	*eSiBu-Tree*	SIM*	a) Seleção dos documentos JSON b) Criação registro na *eSiBu-Tree* c) Visualização do esquema unificado (*skeleton*)
[2]	*Dataset* JSON ou CSV	Migrar para BD Relacional	Organização Hierárquica (grafo)	Grafo dirigido	Modelo relacional	SIM	a) Criação de uma árvore de atributos e mineração de dependências funcionais b) Identificação de entidades sobrepostas c) Geração do esquema físico relacional
[11]	*Oracle* JSON *column*	Consultar coleções de documentos JSON	Organização hierárquica (JSON DOM)	--x--	JSON *DataGuide*	SIM	a) Derivação dos caminhos (*paths*) estruturais hierárquicos de documentos JSON b) Armazenamento de informações estatísticas dos JSON *paths*

sados ao serviço, reduzindo a quantidade de pares chave-valor (por exemplo, para diminuir o tráfego de rede) [5]. Documentos JSON retornados de *Datasets* ou bancos de dados NoSQL, por sua vez, podem conter toda a informação do esquema em uma única instância.

Quanto ao *objetivo*, percebe-se que a maioria dos trabalhos visam a criação de ferramentas para manipular e gerenciar esquemas [5,7,9,12,14]. Dois trabalhos enfatizam o uso de dados JSON em BDs relacionais [2,11] e apenas 2 trabalhos mencionam o uso dos esquemas para integração de fontes de dados distintas [8,14].

Em relação a *abordagem* utilizada no processo de extração, são identificadas três categorias: *a*) organização hierárquica [2,9,11,14], que utiliza alguma estrutura em árvore ou grafo para suportar o processo de extração ou representação de esquemas; b) transformação de modelos [5,7,12], que faz uso de técnicas de *Model Driven Engineering* (MDE) [13] para definir um esquema para os dados JSON. Gera-se um esquema para cada documento JSON e, posteriormente, faz-se a unificação dos esquemas em um único esquema de domínio; *c*) organização baseada em ontologia [8], que aposta em tecnologias da *Web* semântica para realizar a integração de dados.

Pode-se fazer uma correlação entre o objetivo do esquema e a abordagem adotada no processo: propostas visando a criação de ferramentas para trabalhar com esquemas JSON adotam, preferencialmente, alguma técnica de transformação de modelos. Já propostas que visam a integração de dados ou a realização de consultas a um esquema geralmente adotam alguma estrutura hierárquica ou ontologia para representar o esquema final.

A grande maioria dos trabalhos utiliza um modelo de dados intermediário diferente do modelo de dados usado para representar o esquema JSON final. Ele é utilizado geralmente nas etapas iniciais do processo com o objetivo de representar a estrutura de uma única instância de documento JSON. Alguns trabalhos propõem uma estrutura própria, em formato de árvore ou grafo, utilizada para representar a estrutura de objetos aninhados de documentos JSON [2,9,14]. Abordagens baseadas em MDE [5,12] usam metamodelos ECORE [13], tirando vantagem das ferramentas providas por essa tecnologia, por exemplo, para representar metamodelos como diagramas de classe UML. Trabalhos que citam o uso de esquemas JSON adotam, em geral, uma representação própria do esquema no formato JSON, normalmente construções do tipo *chave:tipo_de_dado*. Apenas o trabalho de Kapsammer *et al*. [7] afirma utilizar a especificação JSON *Schema* [6] como modelo intermediário.

Com relação à representação final do esquema para documentos JSON também não há um consenso. Os trabalhos que usam a abordagem de transformação de modelos apresentam o esquema usando algum formato de metamodelo ECORE [5,7,12], normalmente tratado como metamodelo de domínio da aplicação. Outros trabalhos utilizam uma estrutura própria, como é o caso da *eSiBu-Tree* [14] e do JSON *DataGuide* [11], ou geram um esquema físico relacional [2]. Apenas em [9] percebe-se o uso efetivo da especificação JSON *Schema* [12]. Em [8], que trata de integração semântica, é adotada a OWL como modelo final de representação do esquema.

A falta de um esquema de dados explícito (*schemaless*) é uma característica atraente em bancos de dados NoSQL para desenvolvedores de aplicação. Em função disso, a evolução dos dados também é mais fácil porque não há necessidade da evolução

do esquema, fazendo com que diferentes versões de dados possam coexistir. A maioria dos trabalhos apresenta um único esquema ao final do processo, o qual busca representar todo o conjunto de dados existente na fonte de origem. No entanto, percebe-se que alguns trabalhos se preocupam em distinguir as diferentes variações de esquemas que os dados podem ter no banco de dados. Em [12] é proposto o versionamento de esquemas, sendo que as versões de esquemas permitem representar a evolução dos dados no tempo. Em [14] também é gerado um *record schema* para cada tipo de documento JSON, distinguindo as diferenças estruturais presentes na coleção. No entanto, os autores propõem o uso do conceito de *skeleton* para criar uma representação única do esquema para os documentos JSON, a qual contém os principais atributos da coleção, identificados através de uma medida de qualidade. Em [9] são apresentados esquemas únicos por coleção. Além disso, identificam-se atributos obrigatórios e opcionais e utiliza-se um grau de cobertura como medida para definir o grau de similaridade entre dois documentos JSON.

Com relação às etapas de desenvolvimento dos processos de extração de esquemas, percebe-se uma certa semelhança entre os trabalhos. Cada etapa representa desafios que precisam ser tratados e podem ser resumidos em: *a*) selecionar o conjunto de documentos JSON a ser considerado na geração dos esquemas (essa etapa é influenciada pela fonte de dados JSON); *b*) extrair o esquema de cada documento JSON e representá-lo em um modelo; *c*) inferir um esquema de domínio (único), por meio da integração dos esquemas, de cada documento JSON.

5. Considerações Finais

O volume, variedade e velocidade dos dados atuais impulsionou o surgimento de modelos de representação de dados complexos, como o formato JSON, e novas categorias de bancos de dados, como os bancos de dados NoSQL, os quais são fortemente caracterizados pela ausência de esquema. A falta de definição de esquema oferece uma maior flexibilidade, facilita a inclusão de dados não uniformes e a evolução dos dados. Neste cenário, percebe-se atualmente o interesse em extrair esquemas de dados JSON visando o desenvolvimento de ferramentas de gerência de dados ou a integração de bancos de dados distintos. O problema da extração de esquemas JSON é ainda uma questão em aberto na comunidade de banco de dados. Assim sendo, a contribuição principal deste trabalho é a apresentação e comparação de propostas que sintetizam soluções possíveis para este problema, visando servir como um guia para a pesquisa pelos interessados no assunto.

A partir do quadro comparativo apresentado pode-se inferir alguns desafios que precisam ser explorados na busca por soluções mais efetivas. Alguns deles podem estar relacionados ao domínio da aplicação, como, por exemplo:

- Seleção de registros: no contexto de *BigData*, em que coleções possuem um grande volume de dados, pode não ser viável usar todo esse volume em um processo de extração de esquemas. Dessa forma, é necessário definir estratégias para a seleção dos dados;
- Formato de representação de esquemas: o padrão JSON *Schema* ainda se encontra em desenvolvimento. Necessita-se verificar se ele já está em condições de atender as necessidades de representação ou se é melhor usar uma das alternativas propostas ou ainda definir novos padrões;

- Unificação de esquemas: a disponibilização de um único esquema ou vários esquemas que representem as mudanças estruturais nas coleções de dados JSON não tem uma resposta final. Técnicas e ferramentas para fazer *merge*/*matching* de esquemas precisam ser adaptadas para uso com JSON *Schema*.

Referências

[1] **DB Engines**. 2016. Disponível em: <http://db-engines.com/em/ranking_trend>. Acesso em: 07 dez. 2016.

[2] DISCALA, M.; ABADI, D. J. Automatic Generation of Normalized Relational Schemas from Nested Key-Value Data. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA - SIGMOD, 2016. **Proceedings...** New York: ACM, 2016.

[3] GHAREHCHOPOGH, F. S.; KHALIFELU, Z. A. Analysis and evaluation of unstructured data: text mining versus natural language processing. In: INT. CONF. ON APPLICATION OF INFORMATION AND COMMUNICATION TECHNOLOGIES - AICT, 5., 2011. **Proceedings...** Azerbaijan, Baku, 2011.

[4] HAINAUT, J. L. *et al*. The Nature of Data Reverse Engineering. **Data Reverse Engineering Workshop**, EuroRef, Seventh Reengineering Forum, Reengineering Week 2000, Zurich, Switzerland, March 2000.

[5] IZQUIERDO, J. L. C.; CARBOT, J. Discovering implicit schemas in JSON data. **Lecture Notes in Computer Science**, v. 7977, Heidelberg: Springer-Verlag, 2013.

[6] JSON Schema Community. **JSON Schema**, 2016. Disponível em: <http://json-schema.org>.

[7] KAPSAMMER, E. *et al*. User profile integration made easy - Model-driven extraction and transformation of social network schemas. In: ANNUAL CONFERENCE ON WORLD WIDE WEB COMPANION – WWW, 21., 2012. **Proceedings...** 2012.

[8] KIRAN, V. K.; VIJAYAKUMAR, R. Ontology Based Data Integration of NoSQL Datastores. In: INDUSTRIAL AND INFORMATION SYSTEMS – ICIIS, 9., 2014. **Proceedings...** 2014

[9] KLETTKE, M.; STÖRL, U.; SCHERZINGER, S. Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In: BTW. **Proceedings...** LNI.GI, 2015

[10] JSON.ORG. **Introducing JSON**. Disponível em: <http://json.org>. Acessado em: 17 dez. 2016.

[11] LIU, Z. *et al*. Closing the Functional and Performance Gap Between SQL and NoSQL. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA – SIGMOD, 2016. **Proceedings…** San Francisco (Califórnia): ACM, 2016

[12]RUIZ, D. S.; MORALES, S. F.; MOLINA, J. G. Inferring versioned schemas from NoSQL databases and its applications. **Lecture Notes in Computer Science**, v. 9381, p. 467–480, 2015.

[13] STEINBERG, D. *et al*. **EMF - Eclipse Modeling Framework**. 2. ed. Boston: Pearson, 2009.

[14] WANG, L. Schema Management for Document Stores. **The VLDB Endowment**, v. 8, n. 9, p. 922–933, 2015.