Em direção a um Catálogo de Padrões para Arquiteturas de Processamento de Dados em Tempo Real

Alternative Title: Towards a Patterns Catalog for Data Stream Processing Architectures

Osman de O. Lira Junior
CESAR.EDU
Recife, Brasil
Faculdade Projeção
Brasília, Brasil
osman.lira@gmail.com

Jorge Fonseca
Universidade de Pernambuco
Recife, Brasil
jorge.fonseca@upe.br

Kiev Gama
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brasil
kiev@cin.ufpe.br

RESUMO

Como fruto do avanço tecnológico, vários dispositivos informatizados estão se integrando através da internet, e com isso surge um número crescente de aplicações distribuídas que requerem processamento contínuo de um grande fluxo de fonte de dados, geograficamente distribuídas, em volumes imprevisíveis, necessitando obter respostas rápidas para consultas complexas. Essas aplicações, capazes de processar grandes quantidades de informação em tempo útil, são conhecidas como Information Flow Processing (IFP) [1]. Apesar de terem um objeto em comum, estes sistemas se diferenciam em vários aspectos, incluindo a arquitetura, modelos de dados, linguagens de regras e mecanismos de processamento. Recentemente, muito esforço foi colocado na tentativa de definir um background comum para os sistemas IFP. No entanto, estas iniciativas ainda estão muito incipientes e nenhum modelo real, padronizado e unificado foi proposto até agora para descrever e classificar os sistemas de processamento de eventos complexos. A proposta deste artigo é a criação de um catálogo de padrões para capturar os diferentes aspectos de um sistema de IFP e usá-lo para proporcionar uma classificação ampla dos sistemas e mecanismos utilizados fornecendo uma referência para arquitetos de sistemas de informação que tenham como requisito o processamento de dados em tempo real.

Palavras-Chave

Complex event processing, publish-subscribe, stream processing, design patterns.

ABSTRACT

As a result of technological advances, many computerized devices are communicating over the internet, and with that comes a growing number of distributed applications that require continuous processing of large data stream source, geographically distributed in unpredictable volumes, requiring quick answers for complex queries. These applications, able to process large amounts of information in a timely manner, fit in the category of Information Flow Processing (IFP) [1]. Although they have a common object, these systems differ in many aspects, including the architecture, data models, and rules language processing mechanisms. Recently, much effort was put in the attempt to define a common background for the IFP systems. However, these initiatives are still incipient and no real, standardized and unified model has been proposed so far to describe and classify the elements that compose those types of systems. The purpose of this article is to propose a pattern catalog to capture the different aspects of IFP systems, specifically focusing on complex event processing. Such catalog can be used as a reference for architects of information systems that have realtime data processing as a requirement.

SBSI 2016, May 17–20, 2016, Florianópolis, Santa Catarina, Brazil.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; I.5 [**Pattern Recognition**]: Miscellaneous;

General Terms

Design.

Keywords

Complex event processing, publish-subscribe, stream processing, design patterns.

1. INTRODUÇÃO

Vários dispositivos informatizados estão se integrando através da internet, surgindo desta maneira um número crescente de aplicações distribuídas que requisitam processamento contínuo de um grande fluxo dados, geograficamente distribuídas, em volumes imprevisíveis, necessitando obter respostas rápidas para consultas complexas. Essas aplicações, capazes de processar grandes quantidades de informação em tempo útil, são conhecidas como Information Flow Processing (IFP) [1].

Estas exigências têm conduzido os arquitetos de software ao desenvolvimento de um número de sistemas especificamente desenhados para processar a informação como um fluxo contínuo de dados (ou um conjunto de fluxos) de acordo com um conjunto de regras. Apesar de terem um objeto em comum, estes sistemas se diferenciam em vários aspectos, incluindo a arquitetura, modelos de dados, linguagens de regras e mecanismos de processamento. Em parte, essas características existem devido ao interesse de diferentes comunidades científicas na área, cada uma trazendo a sua própria visão do problema, assim como solução e vocabulário específicos [2]. Conforme apontado por Cugola [1], dois modelos surgiram: o modelo de processamento de fluxo de dados [3] e o modelo de processamento de eventos complexos [4].

O modelo de processamento de fluxo de dados descreve o problema dos IFPs como fluxos de processamento de dados provenientes de diferentes fontes, para produzir novos fluxos de dados como saída e trata este problema como uma evolução do processamento de dados tradicional, como defendido pelos SGBDs. Assim, sistemas de gestão de fluxo de dados (DSMSs) têm suas raízes nos SGBDs,

mas apresentam diferenças substanciais. Enquanto SGBDs são projetados para trabalhar em dados persistentes, onde as atualizações são relativamente pouco frequentes, os DSMSs lidam com dados transitórios que são atualizados continuamente.

Por outro lado, os dados do fluxo de modelo de processamento de eventos complexos são itens de informação como notificações de eventos que acontecem no mundo externo, que têm que ser filtradas e combinadas para entender o que está acontecendo em alto nível. Assim, o foco deste é na detecção de ocorrências de padrões particulares de eventos (em baixo nível) que representam os eventos de nível mais alto, cuja ocorrência tem de ser notificadas às partes interessadas. As contribuições para este modelo vêm de diferentes comunidades, incluindo sistemas distribuídos de informação, automação de processos de negócios, sistemas de controle, monitoramento de rede, redes de sensores, e middleware, em geral. A origem deste modelo é influenciada por sistemas de mensageria publish-subscribe[5].

Recentemente, muito esforço foi colocado na tentativa de definir um background comum para os sistemas IFP. Surgiram propostas, principalmente da comunidade de processamento de eventos, onde uma grande discussão sobre o tema acontece desde o surgimento do livro que popularizou o termo processamento de eventos complexos [4][6]. A Event Processing Technical Society [7] foi fundada para promover a compreensão e avanço no domínio de processamento de eventos e para desenvolver padrões. Todos estes esforços e discussões geram uma grande ênfase sobre as possíveis aplicações e usos de sistemas de CEP, bem como na integração com as soluções corporativas existentes. Por estas razões, CEP têm recebido uma grande atenção por parte da indústria, que está rapidamente adotando o termo. No entanto estas iniciativas ainda estão muito incipientes e nenhum modelo que cubra os seus requisitos e sirva de referência para construção de sistemas que processam fluxos de informação foi proposto até agora para descrever e classificar os sistemas de CEP ou mesmo de IFP de maneira geral.

Este artigo traz uma visão mais genérica que o CEP, trazendo foco em IFP, e propõe um catálogo de padrões para capturar os diferentes aspectos de um sistema de IFP. A partir dele visa-se uma classificação ampla dos sistemas e mecanismos utilizados naquele contexto. Através deste catálogo pretende-se fornecer uma referência para arquitetos de sistemas de informação que pretendem construir sistemas que tenham como requisito o processamento de dados em tempo real.

O restante do artigo apresenta os seguintes tópicos: Seção 2 traz a fundamentação sobre padrões de projetos e IFPs; Seção 3 apresenta os trabalhos relacionados; seção 4 apresenta a direção do catálogo proposto; Seção 5 traz as conclusões e trabalhos futuros.

2. FUNDAMENTACAO

2.1 PADRÕES DE PROJETO

Padrões de projeto representam um avanço considerável para a área de orientação a objeto e arquitetura de software, uma vez que disponibiliza um catálogo de soluções reutilizáveis que já foram testadas e provaram ser eficientes para a resolução de problemas semelhantes e recorrentes[15]. O nível de abstração dos padrões de projetos pode variar bastante, mesmo assim, tem características comuns e em diversos casos estão fortemente relacionados uns com os outros. Diversas aplicações e bibliotecas pode fazer uso de um ou mais padrões.Os padrões criam um vocabulário comum para discutir arquiteturas quando nos referimos aos nomes dos padrões nos servindo de guia ou material de referência. Dito isto, devem ser definidos em catálogos, onde são organizadas de acordo com diferentes critérios de atribuição e granularidade. Existem vários catálogos de padrões, como exemplo, podemos citar os padrões de programação orientada a objetos [15], padrões orientado a arquitetura [16], padrões de integração de aplicações [17], padrões de segurança [18], padrões para comunicação [19], e assim por diante.

2.2 IFPs

O termo *Information Flow Processing* (IFPs) se refere a um domínio de aplicação em que os usuários precisam coletar informações produzidas por múltiplas fontes, distribuídas para processá-la de forma oportuna, a fim de extrair novos conhecimentos, logo que a informação coletada se torne relevante [1].

Atualmente, temos aplicações dos mais variados tipos que possuem estas características, como detecção de fraude, monitoramento ambiental, monitoramento de rede [8][9][10]. Como citado por Cugola [1] estas aplicações possuem o requisito de uma análise contínua de fluxos, ou seja, processamento da borda para o centro do sistema, sem a necessidade de a princípio a informação ser persistida.

Estas aplicações degradam muito a performance por conta do alto volume de dados. Diante disto, arquiteturas foram desenvolvidas para solucionar estas características, com tecnologia conhecida como *In-Memory Computing*, *Stream Processing* e *Complex Event Processing* [11].

3. TRABALHOS RELACIONADOS

Há muito esforço na tentativa de definir um *background* comum para os sistemas IFP. A maioria das tecnologias de processamento de eventos complexos tem adotado linguagens baseadas em SQL para consulta de dados e desta maneira usa os operadores da mesma. Atualmente, padrões de apoio como filtros, transformações, padrões de eventos temporais existem, mas, uma classificação de como e quando usar ainda é algo muito incipiente.

Cugola [1] definiu uma classificação e uma extensa pesquisa de diferentes ferramentas. Entretanto, verifica-se na pesquisa que o enfoque está em operadores básicos abstratos ao invés de padrões de mais alto nível. *Esper* definiu uma média de cinquenta padrões de solução [12], porém, com foco em requisitos específicos. Coral8 [13] cataloga dez padrões, mas muito requisitos importantes da arquitetura não são cobertos. A arquitetura de referência proposta pela *Event Processing Technical Society's* (EPTS) [14] chega próximo desta ideia, identificando alguns padrões de alto nível assim como Perera [11] que identifica treze padrões.

Embora mencionem e cataloguem padrões todos os trabalhos deixam de lado um pouco do princípio da finalidade de padrões, que é justamente categoriza-los demonstrando suas relações entre si.

4. CATÁLOGO PROPOSTO

Como mencionado anteriormente, estamos propondo a criação de um catálogo de padrões de alto nível para o modelo funcional proposto por Cugola (mostrado na Figura 1), baseados nos padrões sugeridos pelos trabalhos relatados na seção anterior, mas que serão categorizados para cada componente do modelo, mostrando suas dependências e em qual contexto deve ser aplicado. A arquitetura funcional proposta por Cugola se assemelha ao modelo de referência MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) proposto pela IBM [20]. Ambos trazem alguns

elementos equivalentes entre os dois modelos, como é o caso da base de conhecimento (*Knowledge Base*), e parte do próprio princípio de funcionamento como um todo. No catálogo aqui proposto, tenta-se classificar padrões que podem ser empregados em cada um dos elementos do modelo proposto por Cugola, podendo também ser empregados na construção de arquiteturas que empregam o princípio MAPE-K. O foco com nosso catálogo é demonstrar uma visão aos arquitetos de software que hoje ainda está em aberto para este tipo de modelo.

Por questões de limitação de espaço no artigo, os vários elementos do formato típico — intent, objective, motivation, structure, participants, etc — de *design patterns* utilizado por Gamma et al [15] não serão descritos. O catálogo atual apenas enumera e brevemente descreve os vinte padrões elencados.

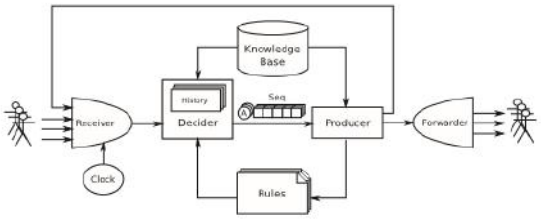

Figura 1. Arquitetura funcional de um IFP[1]

4.1 *Event Receiver Patterns*

Segundo Cugola, os fluxos de informações entram pelo *receiver*, cuja a tarefa é gerenciar os canais que ligam as fontes com o *engine* do IFP. Esta camada é responsável pela implementação do protocolo de transporte adotado pelo *engine* para movimentar as informações pela rede. Nela, teremos os padrões da categoria *Event Receive*, que tem seu contexto ligado a padrões de processamento de preparação do evento e de seus dados e metadados associados para as fases posteriores de processamento de eventos. Os padrões desta categoria incluídos no catálogo são:

1. *Content Identification[14][12]*: Identifica os eventos de entrada em relação a eventos anteriores e tipos de eventos;
2. *Content, Selection[14][12]*: Eventos específicos devem ser selecionados para futura análise ou para algum padrão correspondente;
3. *Content Filter[14][17]*: Filtra todos os eventos que têm alguma propriedade na sua carga;
4. *Content Monitoring[14][11]*: Monitora canais para eventos de interesse;
5. *Content Enricher[14][17][13][11]*: Acrescenta algumas informações com base em eventos anteriores.

Nesta mesma camada (citada acima), temos um elemento *Clock*, que têm como função a criação de elementos de informação periodicamente. Para este elemento temos a categoria *Session clock*, que tem seu contexto ligado a padrões que criem periodicamente relógios para executar elementos de diferentes cenários num determinado tempo. Os padrões desta categoria são:

6. *Real Time Clock[11]:* Implementação de relógios de tempo real que utiliza internamente o relógio do sistema para determinar o *timestamp* atual;
7. *Pseudo Clock[11]:* Implementação de relógios que serão controlados por outra aplicação.

4.2 Event Processing Patterns

Depois de passar pelo *receiver*, os eventos provenientes das fontes externas ou geradas pelo *clock* entram na camada principal de processamento, onde serão analisados de acordo com as regras de processamento armazenadas. Esta camada divide o processamento em duas fases: uma fase de detecção e uma fase de produção. O primeiro é realizado pelo *decider*, que recebe as informações do *receiver*, e analisa com as regras existentes para verificar se satisfaz a condição. As condições satisfeitas passam então para o *producer* para ser executado. Nesta camada teremos os padrões da categoria *Event Decider*. Esses estão relacionados à análise dos eventos para transformá-los em informações pertinentes. Já a categoria *event producer* terá relação com padrões que criam uma nova informação do evento, ou a atualizam de eventos complexos existentes baseados na análise feita na camada de *Decider*. Os padrões destas categorias incluídos como parte do catálogo são:

4.2.1.Event Decider

8. *Analytics [11] [12] [13] [14] :* Usa métodos estatísticos para obter informações adicionais sobre um evento ou conjunto de eventos;
9. *Event Process [11] [12] [13] [14]*: Realiza processos em *payloads* ou em dados do evento. Pode ser uma preparação, análise ou processamento;
10. *Event Tracking[14]:* Realiza o rastreamento dos eventos identificando o seu estado.
11. *Event Scoring[14]:* Classifica eventos e seus dados com base em alguns critérios predefinidos;
12. *Event Rating[14]:* compara eventos e seus dados com métricas para fornecer uma ordenação;
13. *Event Classification[14]:* Comparação de associação de eventos com algum esquema de classificação a que o evento é aplicado.

4.2.2.Event Producer

14. *Event Consolidation[14]:* Combina eventos do mesmo tipo em um evento novo ou combina eventos diferentes em um evento principal ou primário;
15. *Event Composition[14]:* Compõe novos eventos complexos a partir de um existente;
16. *Event Aggregation[14]:* Combina eventos do tipo diferente em um novo evento.

4.3 *Knowledge* Patterns

Cugola [1] menciona o *Knowledge Base* como uma base de conhecimento representada por um repositório estável de informação. Normalmente presente como um repositório *read-only* que acessa dados persistidos tipicamente em tabelas de bancos de dados. Pelo menos dois padrões podem servir de alternativa nesta categoria:

17. *Repository [15]:* Estrutura central que encapsula o acesso a dados da aplicação, funcionando como um repositório de dados;
18. *Data Access Objects [21]:* Fornece uma interface que abstrai o meio de persistência sendo utilizado, encapsulando detalhes do mecanismo de armazenamento de dados sendo utilizado.

4.4 Rule Patterns

Avalia as regras (*Rules)* presentes. O padrão ECA (Event-condition-action), é o que melhor representa o elemento *Rules*:

19. *Event-Condition-Action [22]:* Tipicamente utilizando uma abordagem declarativa, o princípio de uma regra ECA engloba um evento combinado com determinada condição, que irá disparar uma ação correspondente a ser executada.

4.5 Event Forwarding Patterns

O Componente *Forwarding* é o componente responsável por entregar os itens de informações geradas pelo *producer* para os sistemas que solicitaram o processamento da informação. Nesta camada teremos a categoria de padrões *Event Forwarding*, que tem seu contexto ligado a padrões utilizados para descrever o processo de adição de um ou mais novos destinos para um determinado evento. O padrão desta categoria é uma generalização das diversas variações de *Message Router* propostas por Hohpe e Woolf [17]:

20. *Event Routing:* Passa o evento para o serviço adequado com base no contexto do evento (ex: tipo de evento, tipo de dado lista de receptores do evento)

5. CONCLUSÕES E TRABALHOS FUTUROS

Neste artigo, foi apresentado os sistemas de processamento de dados complexos e de como os arquitetos de *software* estão criando iniciativas para tentativa de moldar um modelo unificado e padronizado para descrever e classificar sistemas de processamento de dados complexos. Foi apresentada também a importância dos padrões de projeto nesta arquitetura para gerar uma melhor compreensão do domínio desta tecnologia, facilitando com isso a construção de ferramentas que lidam com esses cenários. Por fim, foi proposta a criação de um catálogo de padrões comuns de alto nível, categorizados para cada componente do modelo funcional apresentado.

A fim de validar este catálogo, como trabalho futuro detalharemos os padrões deste catálogo e o utilizaremos para auxiliar arquitetos de software a identificar arquiteturas e padrões candidatos com base nos requisitos dos sistemas que necessitam construir a fim de validar este catálogo. Com isso, podemos verificar os pontos fortes e limitações da proposta aqui apresentada.

6. REFERÊNCIAS

[1] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. ACM Computing Surveys, 2012.

[2] Bass, T. Mythbusters: Event stream processing v. complex event processing. Keynote speech at the 1st International Confrence on Distributed Event-Based Systems (DEBS'07), 2007.

[3] Babcock, B., Babu, S., Datar, M., Motwani, R.,and Widom, J. Models and issues in data stream systems. In Proceedings of the 21st ACM SIGMOD/PODS Symposium on Principles of Database Systems (PODS'02). ACM, New York, NY, 1–16, 2002.

[4] Luckham, D. C. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2001.

[5] Eugster, P.,Felber,P.,Guerraoui,R.,and Kermarrec,A.The many face sof publish/subscribe.ACM Comput. Surv. 2, 35, 2003.

[6] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.

[7] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/964696.964697.

[8] Broda,K.,Clark,K.,Miller,R.,and Russo,A.Sage:A logical agent-based environment monitoring and control system. In Proceedings of the European Conference on Ambient Intelligence (AmI'09). 112–117, 2009.

[9] Event Zero. http://www.eventzero.com/solutions/environment.aspx. Last accessed 11/2010, 2010a.

[10] Demers, A., Gehrke, J., Hong, M., Riedewald, M.,and White, W. Towards expressive publish/subscribe systems. In Proceedings of the International Conference on Extending Database Technology (EDBT). 627–644, 2006.

[11] Srinath Perera, Solution patterns for realtime streaming analytics. DEBS '15 Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems Pages 247-255, 2015.

[12] Esper solution patterns. http:/www.espertech.com/esper/solution_patterns.php. Accessed: 2015-05-02.

[13] Complex event processing: Ten design patterns. http://complexevents.com/wp-content/uploads/202007/04/Coral8DesignPatterns.pdf. Accessed: 2015-05-02.

[14] A. Paschke and P. Vincent. A reference architecture for event processing. In Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, page 25. ACM, 2009

[15] Gamma, E et al. Design patterns: Elements of reusable object-oriented software. Reading: Addison-Wesley, 1995

[16] F. Buschmann et al., Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, John Wiley & Sons, 2007

[17] Hohpe, Gregor, and Bobby Woolf. "Enterprise integration patterns." 9th Conference on Pattern Language of Programs. 2002.

[18] YODER, Joseph; BARCALOW, Jeffrey. Architectural patterns for enabling application security. Urbana, v. 51, p. 61801, 1998.

[19] VÖLTER, Markus; KIRCHER, Michael; ZDUN, Uwe. Remoting patterns: foundations of enterprise, internet and realtime distributed object middleware. John Wiley & Sons, 2013.

[20] Kephart, Jeffrey O.; Chess, David M. The vision of autonomic computing. Computer, v. 36, n. 1, p. 41-50, 2003.

[21] Alur, Deepak et al. Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies. Sun Microsystems, Inc., 2003.

[22] Paton, Norman W.; Díaz, Oscar. Active database systems. ACM Computing Surveys (CSUR), v. 31, n. 1, p. 63-103, 1999.