DFDを用いた大規模イベントデータの分析支援環境

野村佳秀[†1] 木村功作[†1] 栗原英俊[†1]
山本里枝子[†1] 山本晃治[†1] 徳本晋[†1]

近年，センサーやスマートフォンなどの普及に伴い，多数のイベントデータがリアルタイムに大量に発生している．このような大規模なイベントデータを分析しリアルタイムに処理するサービスを，DFD (Data Flow Diagram)[1] を用いた実装に依存しないフローを使って定義することによって，双方の処理を統合的に記述する手法を提案する．またこの手法を実現する開発環境のプロトタイプによって，実際に分析作業からサービス開発までの作業を分析者，開発者で明確に分離が可能になり，サービス実施までの期間を大幅に短縮できることを示した．

Massive event data analysis and processing service development environment using DFD

Yoshihide Nomura,[†1] Kosaku Kimura,[†1]
Hidetoshi Kurihara,[†1] Rieko Yamamoto,[†1]
Kouji Yamamoto[†1] and Susumu Tokumoto[†1]

In these days, there are a lot of massive event data appeared every day because the networked devices such as smartphone or some kind of sensor become very common. And every servicer intend to analyze the event data and discover the new business rule for real time event processing service. We propose the methodology which integrates such kind of analysis and service development using implement independent DFD (Data Flow Diagram), and develop its prototype for evaluation. It could separate the concern of the developer and analyzer and we succeeded to confirm the prototype tool could reduce actual analysis and service development costs dramatically.

†1 富士通研究所　ソフトウェアイノベーション研究部
Software Innovation Lab., Fujitsu Laboratories Ltd.

1. はじめに

近年，スマートフォンに代表されるネットワークに接続したデバイスやセンサーの爆発的普及，コンシューマ向けサービスのデータの API 公開，また画像や動画などのマルチメディアデータの増加などの流れから，現在全世界に 8000 億 GB 以上あると言われるデジタルデータが，今後 10 年で 44 倍にも増えると言われており[2]，ICT システムが処理の対象とするデータは大規模化してきている．

また，大量に発生する各種のイベントをリアルタイムに処理して，適切なタイミングで人やシステムにフィードバックをかけるイベント処理サービスにより，大量の情報から新しい価値を見出してタイムリーに活用するという高度な情報処理が求められてきている．

2. 大規模イベントの分析処理とリアルタイムイベント処理

本稿では，大規模なイベントの分析処理とリアルタイムなイベント処理を対象としている．大規模なイベントの例としては，鉄道のイベントが挙げられる．関東首都圏では，Suica やモバイル Suica に代表される電子切符がかなりの割合で普及しており，JR 東日本の駅での入場出場時に発生するイベントは，1 日に 2000 万件以上[3] であり，大規模なイベントデータと言える．

従来イベントデータの傾向を把握するには，BI ツールなどの統計分析ツールを利用するのが一般的だが，数テラバイトを超えるような大規模なデータの場合は，文献 4) のように Apache Hadoop[5] などの分散処理エンジンを活用する例が増えている．

文献 6) では，気象データを元にリアルタイムな嵐の予報を行う手法を提案しているが，ここで用いられるルールは事前にデータマイニングツールを使って算出している．また文献 7) ではイベント処理事例が多く提案されているが，ここで用いるルールはクラスタ分析，シーケンス抽出などのマイニング手法を用いて得ることを前提としている．このように，大規模なイベントデータの統計分析処理と，リアルタイムなイベント処理の間には密接な関係がある場合が多い．

一般的にこのようなシステム構築の手順としては，分析対象とするデータや業務に応じて，どのような分析処理が必要なのかを分析者が決定し，それに従って開発者が分析アプリの開発，結果のレポート化を行う．その後，分析結果を元に分析者が有効と思われるサービスのルールを決定し，このルールに従って，CEP (Complex Event Processing) などのイベント処理システムを使って開発者がサービス開発，実施を行う．

表 1　従来のフローの主な違い
Table 1　Key differences of legacy flow model.

特徴	分析処理フロー	イベント処理フロー
データの受け渡し	プロセス内で共有	チャネルを介して送受信
処理の起動	トークンの遷移時	サービス起動時
状態の保持	プロセス毎に処理間で共有	サービス毎に独立して状態を管理

しかし，分析処理およびイベント処理を行うシステムはそれぞれ全く異なるインフラやツールを使って開発を行う必要があり，分析で得られた新しいルールを元にサービスを開発する手間が非常に大きいという課題がある．

3. DFD を使った分析処理とリアルタイムイベント処理の記述

3.1　従来のフローモデル

分析処理，およびイベント処理をグラフィカルに記述する方式としては，フローモデルを使った記述方法がある．分析用のフローと，イベント処理を行うためのフローは，どちらも各種分析処理や CEP 定義間を遷移矢印で繋いだ形になるため，ビューとしては同じように見える．しかし，実行時のモデルを比べると全く異なる．

分析処理のフローは，文献 8) で示されるようにあるきっかけでプロセスインスタンスが開始して初期状態が決まった後，トークンがアクティビティの間を遷移して，アクティブになったアクティビティに対応する処理が順次実行される．この際に利用するデータは，別途プロセスインスタンスの持つ変数，または外部に存在するストレージという形で管理され，基本的にトークン（状態）とデータは直接関係しない．

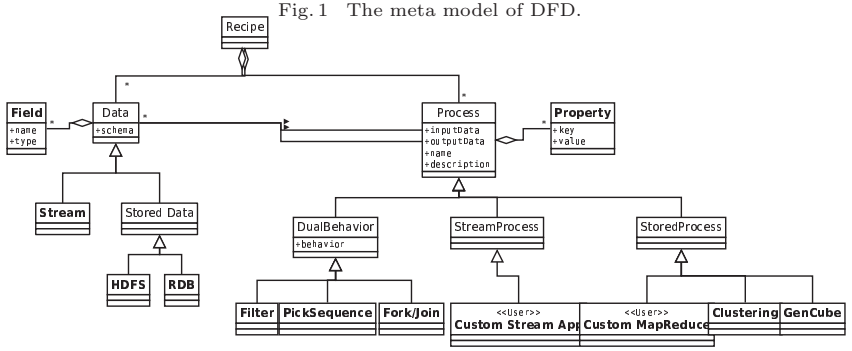
一方イベント処理は，文献 9) で示されるようなモデルで表され，イベントチャネルを介してサービスとして起動しているイベント処理エージェントの間をイベントデータが流れる．各エージェントはそれぞれ独自に状態を持ち，イベントデータによって内部状態を変化させる．

この 2 種類のフローの主な違いをまとめると**表 1** のようになり，これらの基本的な性質の違いから一般的には完全に別の意味を持つフローとして記述する必要があった．

3.2　DFD を使った実装に依存しないフローモデル

そこで我々は，分析処理，イベント処理どちらにも依存しない，DFD (Data Flow Diagram) 形式でそれぞれの処理を統合的に記述する方式を提案する．両者を統合することにより，分析で得られたルールをイベント処理に迅速に反映することを可能とする．

図 1　DFD のメタモデル
Fig. 1　The meta model of DFD.



DFD の基本的な要素は，データとそれに対する処理（プロセス）の依存関係から成り立っており，この解釈については従来の DeMarco の DFD[1] と同様のものとした．DFD では，ワークフローや分析処理などと違い，フローの要素間の実行順序は明確には指定されない．

DFD のメタモデルは**図 1** のように設計した．基本となるのはデータとプロセス（処理）の関係となり，それぞれプロパティやスキーマを持つ．また，データの形式や処理内容に応じてサブクラスを持ち，処理はさらに複数のデータ形式をサポートするもの，特定のデータ形式のみをサポートするものに分かれている．

また，従来分析処理における SQL，イベント処理における Esper[10] の EPL (Event Processing Language) などの DSL (Domain Specific Language) を使って記述されていたフィルタ条件なども，DFD の詳細なプロセスとして展開することで，ある程度の条件であれば DFD のみで記述できる方式を取った．**図 2** は同様のフィルタ条件を，従来の典型的な CEP の定義と，DFD の定義として記述した例である．従来の CEP では，複数の値による絞り込み条件を EPL を併用して記述しなければならないのに比べ，DFD では同様の記述をフィールド a に対する絞り込み条件と，フィールド b に対する絞り込み条件といったより細分化されたプロセスの依存関係として記述できることがわかる．

3.3　DFD を採用するメリット，デメリット

DFD を採用することにより，以下のようなメリットが得られる．

- 分析処理，イベント処理のインフラ，処理方法を意識すること無く，データに対する処理のみを記述することができる．

図 2　従来の CEP 向けの定義と DFD との違い
Fig. 2　Differences between the legacy CEP flow and DFD flow

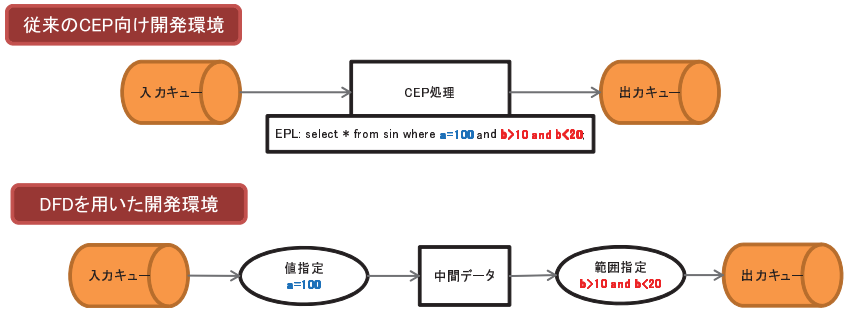
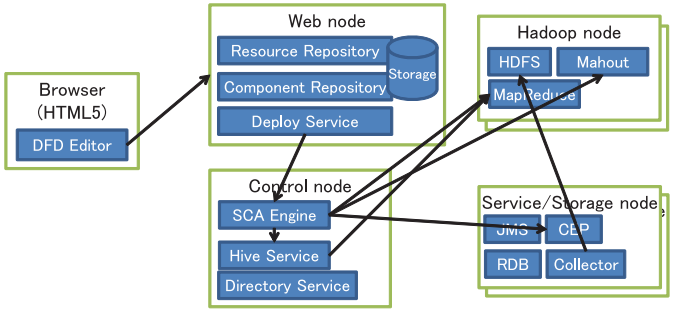


図 3　実行時のアーキテクチャ
Fig. 3　Runtime architecture



- 分析処理で得られたルールを，そのままイベント処理サービスの開発に活かすことができる．
- イベント処理サービスで用いたルールが，どのような分析から得られたのかをノウハウとして蓄積しておくことができる．

分析者，開発者の作業はそれぞれ分離され，並行して実施が可能になる．また，分析結果の解釈からイベント処理サービスルールの定義を全て DFD 上で行うことができ，かつイベント処理サービスの実施が即時可能になるため，サービス実施を迅速に行うことができる．

一方，以下のようなデメリットも考えられる．

実際の動作イメージが掴みにくい． DFD のレベルでは，依存の関係と処理の順序は必ずしも一致しないため，実際の動作は実行時に決定される．そのため，実際に分析処理やイベント処理を実施した際，実行状況を DFD と同じレベルで可視化することが難しい．

コントロールフローの表現． 現状，条件の OR 結合や，条件分岐などのコントロールフローは DFD の要素としてマッピングが難しいことから，意図的に避けている．しかし実際のイベント処理サービスではこのような記述も必要と思われる．

4. ツール試作と評価

4.1 分析支援環境の試作

図 5 が DFD を編集するエディタのイメージである．

フローを定義する分析者は，Web ブラウザから DFD によるデータと処理の流れを記述する．この際，左側のパレットに並んでいる処理部品を適宜フロー上にドラッグ&ドロップで配置しながらフローを定義する．

分析支援環境の試作では，データについては，イベント処理の対象となるストリーム形式と，分析処理の対象となる蓄積データを記述可能とし，プロセスについては，評価に用いたシナリオを実行するために，**表 2** のようにどちらの形式にも対応可能なもの 12 種，またどちらかの形式のみ処理が可能なもの 7 種を部品として用意した．

実行時には，SCA (Service Component Architecture) 定義を用いて，各種コンポーネントの接続を行い，実行エンジンとしては Apache Tuscany[14] を採用した．SCA の定義では Java のコンポーネントのインタフェース間をワイアリングで接続し，その間の通信プロトコルを宣言することによってコンポーネント間が接続される．

図 3 は実行時のアーキテクチャの概要である．ブラウザ上で定義された DFD は Resource Repository 上で管理された後，Deploy Service によって SCA 定義に変換され，Control node 上で実行される．また SCA 定義からは適宜 Hadoop[5] 上の MapReduce ジョブ，Service node 上の Esper[10] が起動，接続され蓄積されたイベントの分析処理，およびリアルタイムなイベント処理サービスが実行される．

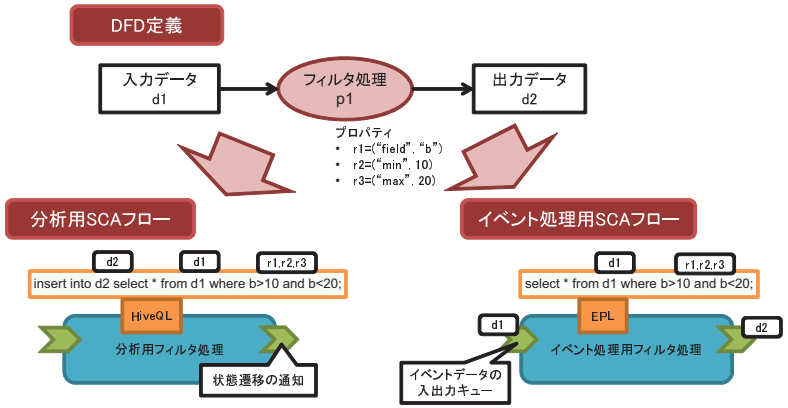
図 4 はフィルタ条件の実行用コンポーネントへの変換例である．入力データ $d1$ をフィルタ条件でフィルタした後，出力データ $d2$ を得るという DFD の定義例となる．

蓄積されたデータの分析処理の場合，フィルタ条件は Apache Hive[13] を使ってクエリを発行するコンポーネントを用いて実行され，その上で動作する HiveQL がプロセスのプロパティを元に生成される．また入出力データ $(d1, d2)$ は，HiveQL 内のデータ名として指定される．

表 2　分析支援環境の試作で用意した DFD プロセス例
Table 2　DFD Process examples of development environment prototype.

プロセス名	対応データ形式	処理概要	利用 OSS
K-means クラスタリング	蓄積	K-means 法によるクラスタリング	Mahout[11]
OLAP	蓄積	多次元分析の実行	Pentaho Mondrian[12]
Suica クレンジング	ストリーム	Suica データのクレンジング（アプリ固有）	-
値ごとの数え上げ	ストリーム／蓄積	指定したフィールドの値ごとのカウント値を計算	Hive[13] / Esper[10]
値指定	ストリーム／蓄積	指定した値のフィールドを持つデータを抽出	Hive/Esper
値範囲指定	ストリーム／蓄積	指定した範囲内の値のフィールドを持つデータを抽出	Hive/Esper
イベント列抽出	ストリーム／蓄積	特定の順序で出現したデータの列を抽出	Hive/Esper
キューブ生成	蓄積	OLAP キューブを生成	Pentaho Mondrian,Hive
距離計算	ストリーム／蓄積	データに含まれる 2 点の位置を表すフィールドから 2 点間の距離を計算	-
クーポンの送信	ストリーム	入力データの SuicaID と結び付けられているユーザにクーポンを送信（アプリ固有）	-
区間ごとの数え上げ	ストリーム／蓄積	指定したフィールドの値の度数分布を作成	Hive/Esper
クラスタ ID 追加	蓄積	クラスタデータに ID を付与	Hive
クラスタの表示	蓄積	クラスタをグラフにプロットし可視化	Mahout
差分計算	ストリーム／蓄積	指定した 2 つのフィールド間の差を計算	Hive/Esper
データ結合	ストリーム／蓄積	2 つの入力データを，あるフィールドをキーとして結合	Hive/Esper
特定フィールド抽出	ストリーム／蓄積	入力データのうち，指定したフィールドだけを抽出	Hive/Esper
日時→時刻	ストリーム／蓄積	日時形式のフィールドから時刻を取り出し新たなフィールドを追加	Hive/Esper
フローの分岐	ストリーム／蓄積	データはそのままで，フローを 2 系統に分岐	Hive/Esper
ランキング取得	ストリーム／蓄積	データを指定したフィールドの値で降順でソート	Hive/Esper

図 4　DFD から SCA 実行定義への変換イメージ
Fig. 4　Conversion image from DFD flow to executable SCA definitions.



一方，リアルタイムなイベント処理の場合，フィルタ条件は Esper エンジンをサービスとして起動するコンポーネント上で実行される．この場合，フィルタ条件に対応する EPL 定義が生成され，入出力データ $(d1, d2)$ に対応するキューがコンポーネントの入出力として接続される．

4.2　分析支援環境の評価

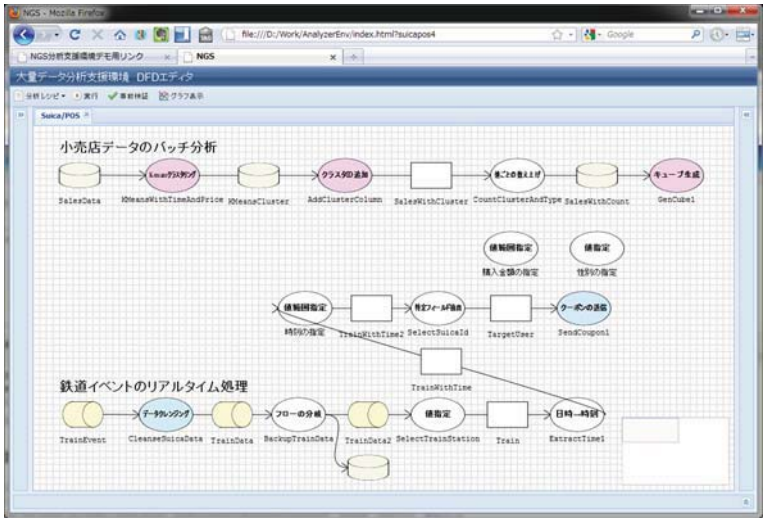
今回試作したツールを利用した場合において，分析者と開発者の作業量がどのように変化するかを，小売店の売上データ，および電車の乗降データを使った分析，イベント処理サービスを従来の手法，および試作ツールをそれぞれ用いて同一の分析者，開発者が開発することで評価を行った．図 5 がその DFD 記述の例である．

この例では，上半分のフローでまず POS データを Apache Mahout[11] を使ってクラスタ分析し，その結果をさらに Pentaho[12] を使ってドリルダウン分析を行っている．分析の結果，下半分のフローで最終的に効果が見込めると判断した顧客層に対して，改札から出た時点でリアルタイムに店舗からクーポンを発行する，というバッチ分析およびリアルタイムなイベント処理サービスを記述している．

まず従来のように，分析アプリとイベント処理サービスを別個に開発する場合，手法分析者の作業は主に下記の 3 つであり，実質約 2 週間ほどの作業期間がかかった．

(1)　分析手順の定義（約 3 日）

図 5　DFD による鉄道イベントと POS データを使ったサービス定義の例
Fig. 5　DFD flow example of service using train event and pos data.

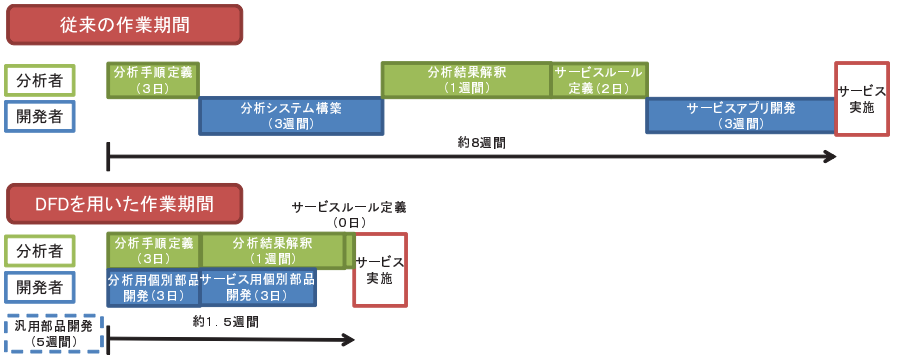


(2)　分析結果の解釈（約 1 週間）
(3)　サービスルールの定義（約 2 日）

開発者は，分析システムおよびサービスアプリの開発作業が必要となり，全体で約 30k ステップの Java コードを約 6 週間かけて開発した．分析作業と開発作業は基本的に逐次的に行うことになるため，全体としては約 8 週間の作業期間がかかった．

一方，今回試作した分析支援環境で実施した場合，開発者の作業としては，分析，サービス実施用の DFD 向けの部品機能の開発が必要だが，今回利用した汎用部品にあたる部分が約 32k ステップ，アプリ固有部品が約 3k ステップとなり，アプリ固有部品の開発は約 1 週間で終えることができた．全体のコード量が従来よりも増えているのは，汎用化のためのテンプレートや SCA 向けのラッピングなどが追加されているためである．開発した部品は表 2 に示したもので，アプリ固有と記載されているもの以外は汎用部品となっている．またこの部品開発は分析者による分析作業と並行して行うことができるため，汎用部品を事前に用意しておくことが可能であれば，**図 6** に示すように分析を開始してからサービス実施までの作業時間は約 1.5 週間に短縮することが可能となった．またこれにより，短期間に改善

図 6　作業期間の比較
Fig. 6　Differences of the man-hour between the legacy and DFD.



サイクルを回していくことが可能になった．

4.3　今後の課題

3.3 で挙げたように，純粋な DFD モデルでは表現しにくいコントロールフローなどへの対処方法の検討や，DFD と動作時コンポーネントとのモニタリングのトレーサビリティ確保，本来の DFD 記述[1] で対応している階層的なモデルへの対処など，より実践的なモデルへの対応強化は今後の課題である．また，より幅広い事例や業界で共通的に利用できる各種コンポーネント部品の開発，拡張方式の汎用化などについても順次進めていく．

評価についても，より多くの事例での記述性の評価，汎用化した部品の実行性能の比較などを行っていく必要がある．また今後の方向性として，文献 15) のような形式記述言語への変換技術を応用した DFD のモデル検査による仕様上の不具合の早期発見などが考えられる．

5. 関 連 研 究

DFD から UML モデルへのモデル変換研究は多く行われている．文献 16) に多くの手法が比較されているが，ほとんどはデータとその操作を静的なクラス図に変換する手法であり，一部はシーケンス図，ユースケース図への変換[17] も提案されているが，SCA などの具体的な実行定義に変換する，といった研究はまだ行われていない．

文献 18) では，MDA (Model Driven Architecture) 技法を使って，同一の PIM (Platform Independent Model) である独自の DSL から PSM (Platform Specific Model) である異な

るルールエンジン，CEP エンジンの定義を生成する手法を提案している．文献 19) では，フロー実行用の BPEL にイベント処理用の定義を統合する仕様を提案している．しかし，CEP から入力したイベントを元にプロセスの起動，アクティビティの遷移などを行う仕様であり，CEP 向けの EPL 等は従来同様に別途定義する必要があり，EPL を含む抽象モデルを提案するものではない．文献 20) では，ワークフロー，ビジネスルール，イベント処理をメタモデルを介して統合する手法が述べられており，ある条件に合致する特定のワークフローの処理をメタモデルレベルでルールや CEP に置き換える，といったアプローチを取る．しかし，本研究のように統一的なモデルは想定していないため，ルールを共通的に利用することはできず，また同一の記述から異なる実行方式の実装を扱うといった用途には利用できない．

6. ま と め

大規模なイベントデータの分析処理およびリアルタイムなイベント処理サービスを，DFD を用いた実装に依存しないフローを使って定義する手法を提案し，そのプロトタイプを開発することで，実現可能性を検証した．またこれにより，分析作業からサービス開発までの作業を分析者，開発者で明確に分離することが可能になり，分析からサービス実施までの期間を大幅に短縮できることを示すことができた．

参 考 文 献

1) DeMarco, T.: *Structured Analysis and System Specification*, Prentice Hall PTR, Upper Saddle River, NJ, USA (1979).
2) Gantz, J.: As the Economy Contracts, the Digital Universe Expands, Technical report, IDC Digital Universe (2009).
3) 初瀬雄一，真野明子，永瀬秀彦：高性能，高信頼の Suica システムの実現，情報処理学会デジタルプラクティス， Vol.1, No.3, pp.121–128 (2011).
4) Lai, C.-F., Chang, J.-H., Hu, C.-C., Huang, Y.-M. and Chao, H.-C.: CPRS: A cloud-based program recommendation system for digital TV platforms, *Future Generation Computer Systems*, Vol. 27, No. 6, pp. 823 – 835 (online), DOI:10.1016/j.future.2010.10.002 (2011).
5) : Apache Hadoop, Apache Foundation (online), available from ⟨http://hadoop.apache.org/⟩ (accessed 2011-08).
6) Li, X., Plale, B., Vijayakumar, N., Ramachandran, R., Graves, S. and Conover, H.: Real-time Storm Detection and Weather Forecast Activation through Data Mining and Events Processing, *Earth Science Informatics*, Vol.1, No.2, pp.49–57 (online), DOI:10.1007/s12145-008-0010-7 (2008).
7) Hinze, A., Sachs, K. and Buchmann, A.: Event-based applications and enabling technologies, *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, New York, NY, USA, ACM, pp.1:1–1:15 (online), DOI:http://doi.acm.org/10.1145/1619258.1619260 (2009).
8) Salimifard, K. and Wright, M.: Petri net-based modelling of workflow systems: An overview, *European Journal of Operational Research*, Vol.134, No.3, pp.664 – 676 (online), DOI:10.1016/S0377-2217(00)00292-7 (2001).
9) Sharon, G. and Etzion, O.: Event Processing Network - A Conceptual Model, *VLDB, ACM*, (online), DOI:10.1.1.98.2802 (2007).
10) : Esper: Complex Event Processing Engine for Java, EsperTech (online), available from ⟨http://www.espertech.com/⟩ (accessed 2011-08).
11) : Apache Mahout, Apache Foundation (online), available from ⟨http://mahout.apache.org/⟩ (accessed 2011-08).
12) : Pentaho BI Suite, Pentaho (online), available from ⟨http://www.pentaho.com/⟩ (accessed 2011-08).
13) : Apache Hive, Apache Foundation (online), available from ⟨http://hive.apache.org/⟩ (accessed 2011-08).
14) : Apache Tuscany, Apache Foundation (online), available from ⟨http://tuscany.apache.org/⟩ (accessed 2011-08).
15) Ibrahim, R. and yen, S.Y.: Formalization of the data flow diagram rules for consistency check, *CoRR*, Vol.abs/1011.0278 (2010).
16) Jilani, A. A.A., Usman, M. and Nadeem, A.: Comparative Study on DFD to UML Diagrams Transformations, *CoRR*, Vol.abs/1102.4162 (2011).
17) Fries, T.P.: A framework for transforming structured analysis and design artifacts to UML, *Proceedings of the 24th annual ACM international conference on Design of communication*, SIGDOC '06, New York, NY, USA, ACM, pp.105–112 (online), DOI:http://doi.acm.org/10.1145/1166324.1166350 (2006).
18) Paschke, A., Kozlenkov, A. and Boley, H.: A Homogeneous Reaction Rule Language for Complex Event Processing, *CoRR*, Vol.abs/1008.0823 (2010).
19) von Ammon, R., Ertlmaier, T., Etzion, O., Kofman, A. and Paulus, T.: Integrating Complex Events for Collaborating and Dynamically Changing Business Processes, *ICSOC/ServiceWave Workshops*, pp.370–384 (2009).
20) Dohring, M., Karg, L., Godehardt, E. and Zimmermann, B.: The Convergence of Workflows, Business Rules and Complex Events - Defining a Reference Architecture and Approaching Realization Challenges., *ICEIS (3)* (Filipe, J. and Cordeiro, J., eds.), SciTePress, pp.338–343 (2010).