1B1-R-3-3

Kachako: 誰でも使える全自動自然言語処理プラットホーム

Kachako: a fully automatic natural language processing platform for everyone

狩野　芳伸[*1]
Yoshinobu Kano

[*1] 科学技術振興機構　さきがけ
PRESTO, Japan Science and Technology Agency

Automation is a key feature allowing users to exploit expert technologies without knowledge and skill. Our platform, Kachako, is a thoroughly automated system for natural language processing. Together with our compatible natural language processing toolkit, users can achieve their tasks very easily as Kachako automates installation, workflow creation, workflow execution and result analysis with its GUI. When users have user accounts on remote servers, Kachako can deploy a workflow in a scalable way either as a Hadoop batch processing or a distributed web service. Kachako is compliant with standards such as UIMA in order for developers to reuse a part of the system. Kachako is freely available from our website.

1. はじめに

電子化されたテキストに対する自然言語処理の応用範囲は、電子化テキストを用いるあらゆるアプリケーションといえる。その潜在的なユーザは商用から学術目的、テキストマイニングから教育学まで幅広い。しかしながら、ユーザにとって自然言語処理技術の利用は必ずしも容易とはいえないのが現状である。

たとえば、Wikipedia の全データから特定の種類の名詞を抽出し列挙したいとする。そのためにユーザはどのような作業が必要であろうか。必要なツール群を探し、コンパイルし、組みあわせて大規模処理をすればよい。すればよいが、その過程にはさまざまなトラブルがつきもので、プログラミングのスキルや専門的知識が必要となる。こうした過程は本当にユーザが対処すべき事項なのだろうか。我々は、こうした過程のほとんどは「自動化」が可能だと考える。徹底的な全自動化を行えば、目的の達成にとって本質的なわずかなユーザ作業だけが残るのではないか。

システムを提供する側があらゆるユーザの個別の要求を満たすウェブサービスを展開できれば、全自動化が達成できたといえるが、残念ながらこれは不可能である。まず、システム側が提供できる計算資源には限界がある上、常時展開しメンテナンスしてゆくのは有償であっても難しい。また、個々のユーザの目的は大なり小なり異なっており、それぞれに何らかのカスタマイズが必要である。固定サービスの提供では対応しきれない。

一方で、オープンソースコミュニティの発展と共に、最先端研究の成果から商用レベルの基盤的ツールまでもが無償で利用可能となってきた。また、計算資源のコストは低下し続けており、今や誰でも安価なクラウドシステムを借りることができる。我々はこの状況を生かし、ユーザが「実行のための計算資源（サーバ等）を用意して指定」「入力データを指定」「最終段で実行してみたいツールを指定」するだけで、他のあらゆる処理を全自動化するシステム、Kachako を開発した。Kachako プラットフォームは、ツールの選択・組合せ・並列分散展開実行・視覚化・評価までを徹底的に自動化する。プラットフォームに加え互換ツール群として 100 近い自然言語処理のコンポーネントを配布しており、プラットフォームに統合されている。プラットフォームと互換ツール群をポータブルな形で配布し、ユーザの指定した任意の計算資源上でインストールから大規模処理まで全自動実行を可能にした。

本稿ではKachakoの技術的詳細を述べるが、KachakoのGUI（Graphical User Interface）を使えばユーザはそのような詳細を知る必要はなく、プログラミング作業も不要である。このようなシステムは関連する広汎な技術要素を調和的に統合してはじめて可能な実装であるが、単に統合された巨大システムというだけでなく、技術レベルの異なるユーザの要求に対応できるよう様々な粒度でモジュール化している。モジュール類をJava, UIMA, Hadoop, Ivy, JMSといった各種の標準に準拠させることで、既存のツールの再利用性も同時に最大化するよう考慮して設計した。Kachakoはウェブサイトより無償で配布[1]している。

2. 背景

テキストの自然言語処理ができる統合システムは GATE (Cunningham et al., 2002)、言語グリッド(Ishida, 2006)、U-Compare (狩野ほか, 2008)、TETDM (砂山ほか, 2011) などいくつか提案されてきたが、Kachako のように自動化・再利用性・スケーラビリティを同時に十分に満たすものはなかった。

Kachako は、UIMA (Unstructured Information Management Architecture)に準拠している。UIMA は様々な企業・研究機関で利用されている相互運用性のための枠組みで、メタデータの規定や API を提供している。テレビ番組のクイズ王に勝利した IBM の Watson システムでも用いられた。実装は Apache UIMA としてオープンソースで提供されている。UIMA の実行単位はコンポーネントと呼ばれ、コンポーネントを組み合わせて実行可能な UIMA ワークフローを作成する。実行時のデータ構造は CAS と呼ばれる汎用構造に統一されている。他に、データ型定義の形式やコンポーネントのメタデータ記述形式などを提供している。ウェブサービスの形式としては Apache ActiveMQ 上で展開される UIMA-AS サービスが用意されている。

ライブラリの依存関係解決には Apache Ivy を用いた。Ivy は Java ソースコードのビルドに用いられるツールで、ウェブ上にリポジトリを設置しライブラリの依存関係を記述して配置することができる。同様のツール Apache Maven 形式にも対応している。

連絡先：狩野　芳伸，科学技術振興機構/国立情報学研究所. 〒101-8430 東京都千代田区一ツ橋 2-1-2 国立情報学研究所. E-mail: kano@nii.ac.jp or kano@kachako.org

[1] http://kachako.org/ Java 7 の一般向け自動配布版 Java 7 update 6 のリリース（2012 年後半に予定）後に一般公開。Beta 版テストに興味がおありの方はご連絡ください。

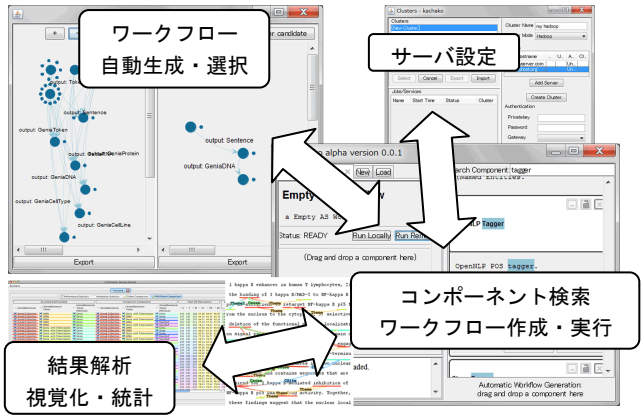

図 1. 典型的な GUI 操作の流れとスクリーンショットイメージ

3. Kachako: 自然言語処理の全自動化システム

3.1 プラットフォームの自動実行と GUI

Kachako プラットフォームは Java で実装されており、Java 7 の動作する環境であればどこでも実行可能である。そこで必要な実行ファイル群をウェブ上に配置し、これを自動的にダウンロード・キャッシングしインストールから起動まで自動化する起動ツールを配布している。起動ツールはクリック動作のみで実行できる。一旦ダウンロードされればオフラインでも起動できる。実行ファイルが更新された場合は自動的に差分をダウンロードし更新する。

Kachako プラットフォームを起動するとデフォルトでは GUI が表示される。図 1 に GUI で完結する典型的なユーザ操作の流れを、図 2 にシステム動作を示す。Kachako はコマンドラインモードも提供している。ワークフローを指定して実行することで、GUI なしで利用できる。標準入出力コンポーネントも提供しており、いわゆるコマンドラインツールとしても実行できる。

3.2 オープン分散コンポーネントリポジトリ

コンポーネントのライセンスが実行ファイルの配布を許すもので、かつポータブルな実装であれば、ウェブ上に適切にパッケージングして配置することでコンポーネントのインストール作業を自動化できる。この点は前節のプラットフォームの自動実行と共通であるが、コンポーネントの数が多くなるにつれその実行ファイルの容量が膨大になるという問題が生じる。コンポーネントリポジトリを効率的な転送が可能な形で別途規格化すれば、プラットフォームとは独立に配布できる。

Kachako の提供するコンポーネントリポジトリに加えて、ユーザは第三者が公開したリポジトリを自由に追加して利用できる。ユーザは単に追加リポジトリの URL を指定するだけでよい。

(1) メタデータと実行リソースの分離、依存関係記述

我々のリポジトリ設計における目標は、ファイル間の依存関係を自動的に解決し、かつファイルを効率的に共有転送することである。Kachako では Ivy のライブラリ依存関係解決部分を利用した。Ivy リポジトリ内のリソースは一意に識別できるため、コンポーネントの実行に必要なリソースを依存関係によって記述すればリソースの共有が可能である。またリソースはローカルディスク等にキャッシュされるため、転送も効率的に行われる。

UIMA では、XML で記述されたメタデータ群と、実行に必要なバイナリ等リソース群は分離されている。Kachako ではワークフロー作成まではメタデータのみを用い、実行に必要なリソースは実行時にはじめて必要に応じて取得するように設計し、ファイル転送の効率化を図った。そのために、リポジトリ自体もメタデータと実行リソースに分離し、メタデータから実行リソースへ依存関係のリンクを記述している。実行ファイルは Java クラスパスにロードすることを想定しているが、非 Java 実行ファイルのために相対パスベースの位置指定を提供しポータビリティを確保した。

(2) コンポーネントの検索

Kachako は登録されたリポジトリのメタデータから UIMA のコンポーネント記述ファイルを収集し、収集されたファイルの各フィールド値に対し全文検索機能を提供する。ユーザが興味のあるコンポーネントを見つけることができるよう、Kachako の GUI はフリーワードの、あるいはフィールド名を指定してのリポジトリ内コンポーネント検索機能を提供している。

3.3 ワークフローの自動組み合わせ生成と評価

(1) コンポーネントの作成と組み合わせ

ソフトウェアの再利用性において、二つ(以上)のツールを組み合わせて利用できるかどうかは核心的な問題である。ツールの入出力定義が文書に記述されていたとしても、そこから組み合わせが可能かを判断するには該当分野の専門知識が必要である。さらに、仮に二つのツールが組み合わせ可能な「はず」だとわかったとしても、実際には後述のフォーマットの形式的互換性や意味的な互換性といった技術的な問題が残る。最悪の場合、ユーザがツールの再実装を必要とすることも多々ある。このような互換性・相互運用性の問題は、ユーザの目的達成にとって本質的な問題ではなく、自動化の余地がある。

組み合わせの問題を整理すると、形式的な互換性・意味的な互換性・実行単位の記述の三つに整理できる。形式的な互換性とは入出力データのフォーマットの互換性であり、これは共通フォーマットを決めれば解決できる。形式的な互換性なので、形式が違っていてもたいていの場合は損失なく相互に変換できる。我々は UIMA を採用することで形式的互換性を担保している。

意味的な互換性はデータ型の問題である。たとえば XML を共通形式として採用したとしても、使用する XML タグセット(DTD や Schema)が一致しなければ組み合わせても意味をなさない。我々はテキストの自然言語処理に用いられるデータ型について階層化した型定義を行い互換化してきた。これは終わりのない問題であり、常に型の追加と整理が必要になる。

実行単位の記述は見過ごされがちであるが再利用性向上のためには非常に重要である。既存のツールはそのまま UIMA コンポーネント化するのではなく極力小さな機能単位に分割したほうが再利用の可能性が高まる。しかし API レベルにまで分割してしまうと、入出力条件が複雑になり組み合わせの可能性が低下する上、入出力条件が機械的に判断可能でないことが問題となる。例えば入出力が String 同士であっても意味のある組み合わせが可能だとは言い難い。我々は、できるだけ小さな機能単位に分割しつつ、データ型のリストのみで入出力を指定できるような形に切り出したものをコンポーネント化すべきと考える。

スケーラビリティは再利用性に加えて考慮すべき問題である。言語処理の多くのツールは、入力テキスト中の一部しか一度に利用していない。入力を情報の独立したブロックに分割できれば、単純な並列化によりスケーラビリティを得られる。例えば異なる文書は、構文的にも意味的にも独立していることが多いから、文書単位で分割して CAS に格納するのが妥当である。

Kachako ではこうした条件を満たした互換 UIMA コンポーネントを作成し多数公開している。詳細は Kachako のウェブサイトを参照されたい。

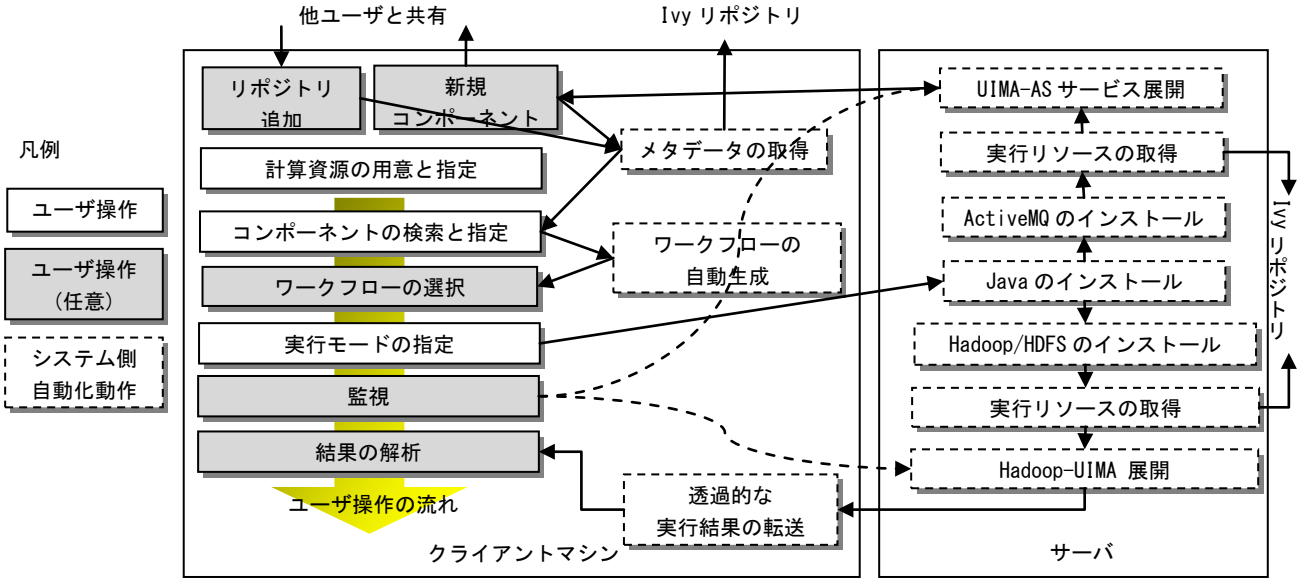

図 2. ユーザの操作と自動化システム動作の概念図

(2) ワークフローの自動生成

コンポーネントが前節で記述したように実装されていれば、入出力条件を用いてコンポーネントのすべての可能な組み合わせを理論的に計算できる。組み合わせは一般の有向グラフになるが、ほとんどの場合ユーザはループをもつワークフローは使用しないため、Kachako のワークフロー自動生成 GUI ではループを削除し DAG（有向非巡回グラフ）にした上で表示している。ワークフロー構成 GUI ではループを含めドラッグ＆ドロップ操作により任意のワークフローを作成できる。

すべての組み合わせを表示すると膨大な数になるため、ワークフロー自動生成時にはユーザに出力コンポーネントを指定させる。最終段の出力コンポーネントはユーザの目的と密接な関係があるから、ユーザは興味あるワークフローを取得できる。それでも多数のワークフローがありえるため、「おすすめワークフロー」をトグルして一つずつ提示する画面を提供し、最短動作として選択するだけで即座に実行することもできるようにした。

(3) 自動組み合わせ比較と評価

同種の出力を得られる多数のワークフローが生成できるとなると、その中から最適のもの選ぶのが次の問題になる。多くの言語処理ツールの振る舞いは入力によって変わり、ほとんどの場合開発者自身にも正確な予測はできない。そのため、実際の入力データでサービスを実行しないことには選びようがない。

多数のワークフローを別個に実行すると時間がかかるが、組み合わせ生成では組み合わせの一部を共有する傾向があるため、我々はこれを利用して重複する結果を共有しつつ全体を物理的には単一の UIMA ワークフローとして実行する、仮想ワークフロー機構を提案した(狩野ほか, 2008)。しかし、ユーザはこの機構を理解しないとワークフロー作成が難しかった。Kachako ではすべてを自動化するため新たな機構を設計し実装した。この機構では既存のコンポーネント実装を修正する必要はなく、コンポーネントからみると仮想的なワークフローの中で実行されているかのように入出力が自動的にフィルタされて受け渡される。ワークフロー生成 GUI で組み合わせのままワークフロー群を選択するだけですべての組み合わせを効率的に計算する。

さらに、同じデータ型の出力同士を自動的に比較できる機構を実装し、比較のメトリックをコンポーネントとして追加できるようにした。いわゆる正解つきコーパスを読み込んで比較すれば、F 値のような統計的な評価値が得られる。

3.4 ワークフローの自動実行・大規模処理と監視

Kachako にはローカル、バッチ、およびリスナの三つの展開モードがある。計算資源の利用上の制約が複雑であっても、多くの場合これらのモードの組み合わせで対応することができる。

(1) ローカル自動実行

ローカルモードでは GUI（あるいはコマンドラインモード）を実行中のマシンでワークフローを実行する。テスト用途に適した最も簡易な実行方法であり、高性能サーバを用いれば大規模処理でも一台で完結しうる。GUI を実行中ならばすでに Java 7 が実行可能な環境なので、あとはシステムが指定されたワークフローの実行をすれば自動実行が実現できる。

まず、前節のコンポーネントリポジトリが正しく構成されている前提で、ワークフローを構成するコンポーネント群の設定ファイルから、実行に必要なリソース群を取得する。コンポーネント毎に別々の Java クラスローダを用意し、実行時にコンポーネントを超えて実行リソースを共有しないことでライブラリのバージョン競合の回避を可能にしている。これは、他のモードでも同じである。ユーザの操作によって作られたローカルにしかないリソースは、自動的に収集されクラスパス上に配置される。

ワークフローの実行は UIMA-AS サービスとして行う。まず ActiveMQ を自動展開し、次にワークフローをサービスとして展開、これを呼び出して実行を完了する。デフォルトでは、実行結果を UIMA XMI 形式で自動保存する。また、実行に用いた設定を後で回復し再利用できるよう、必要な情報も保存する。

(2) リモート自動実行

バッチモードとリスナモードは、GUI とは別のサーバで実行される。同一サーバを指定して実行することも可能であるが、GUI はノートパソコン等で、実行はサーバで行うのが一般的であろう。

リモート実行では Linux ベースの任意のマシンを利用できるよう、極力前提条件を減らす設計を行った。要件は、ユーザが計算資源を利用できるアカウントを持つことと、計算資源が SSH 経由でアクセス可能で、計算資源から HTTP 経由でインターネット接続ができることである。Amazon EC2 のようなクラウドシステムや、研究室の管理するサーバなどさまざまなシステムが利用できる。サーバへのログインは通常、SSH 鍵認証の使用を前提としている。クライアントマシンとリモートサーバ間の通信やファイル転送は、すべて SSH 越しに行われる。管理者などがアカウン

トを設定し Kachako で読み込み可能な形で提供すれば、計算資源を準備し指定する部分のユーザ作業も省略できる。ユーザ自身が指定するのも簡単で、Kachako のサーバ設定 GUI でサーバ名とアカウントを登録すればよい。登録したサーバはシステムにより利用可能かどうか状態の検査が行われる。

root 権限でのアクセスが必要だと、サーバ管理者の作業が必要で自動化の妨げとなる。一般ユーザ権限を前提としたとき問題なのは、ファイルシステムへの書き込み権限である。NFS のような共有ファイル･システムはそのアクセス速度から全体のボトルネックとなりうるため、ローカルディスクかつユーザ権限で書き込み可能なディレクトリを探索するようにした。具体的にはまずユーザのホームディレクトリを検査し、条件に合わない場合は OS による自動的な削除を防止した上で一時ディレクトリを使用する。

Kachako は、実行環境として Java SE 7 を用いるため、Kachako はまず Java を上記ディレクトリに自動インストールする。

(3) Hadoop を用いたリモートバッチ処理

バッチモードでは Apache Hadoop および HDFS を利用し、大規模処理に対応する。Hadoop と HDFS はマスタ/スレーブ構成をとり、それぞれ JobTracker/TaskTracker（Hadoop）・NameNode/DataNode（HDFS）と呼ばれている。Kachako のサーバ設定 GUI ではこれらに割り振るサーバを簡単に指定でき、システムは指定に従って Hadoop/HDFS を自動インストールし構成する。サーバの動的な追加や削除も可能である。

ワークフローの実行は Hadoop/HDFS 上で行う。Hadoop/HDFS 上でタスクを実行するには特殊な API を用いた実装が必要であるが、自動化と再利用性を最大化するためには、任意の UIMA コンポーネントが実装を変更することなく利用できることが望ましい。UIMA コンポーネントの多くは、CAS をうけとり CAS を返すというパイプライン処理を前提としている。つまり、ほとんどのコンポーネントは Hadoop の MapReduce のうち Map だけで事足りる。検索エンジンのインデックス作成など Reduce 演算が必要なものは、別途 UIMA/Hadoop 対応の特別なコンポーネントとして実装し提供していく予定である。

Kachako は任意の UIMA コンポーネントを、HDFS 上で XMI を読み書きするコンポーネントの組ではさんで自動的にワークフローを構成することで、コンポーネントの実装を変更することなく HDFS 上で Hadoop の Map プロセスとして自動実行している。

コンポーネントによっては大きな計算資源を必要とすることがある。ワークフロー全体を 1 台で実行できないような場合のために、ユーザ設定に応じワークフロー内のトップレベルの各コンポーネントを別個の Map プロセスとして順に実行できるようにした。

UIMA コンポーネントの実行はローカルモードと同様に自動化できる。ローカルにしかないリソースはローカルマシン上で Ivy リポジトリを作成し、SSH 接続を介してサーバに転送する。

ワークフローの実行結果は一旦 HDFS 上に保存される。実行結果が大規模であっても、最後に人間であるユーザが直接扱えるデータ量は限られている。であれば、統計的な結果が必要な場合はリモートで統計処理をしてから転送するのが効率的であるし、視覚化機能などで個々のデータを確認するならばその都度転送すればよい。そこで、実行結果はリストのみ転送し、実ファイルは必要に応じて転送するようにすることでユーザがローカルおよびリモートの結果を透過的に扱えるようにした。

大規模処理では実行終了までに長時間を要し、不測の事態が生じがちであり監視機能は重要である。監視におけるユーザの主な関心事は、実行ジョブの生死と進行状況であると考える。Kachako の GUI ではこれらの情報をリアルタイムで表示するとともに、より詳細な監視情報を Hadoop から SSH 経由で転送しユーザがボタンをクリックするだけで表示することができる。

(4) UIMA-AS を用いたリモートウェブサービス展開

リスナ、すなわち待ち受けサービスが必要な状況はいくつか想定しうる。Kachako では再配布可能なソフトウェアを基本としているが、ライセンス等の問題で実行ファイルを配布したくない状況は商用を中心に容易に想像できる。また、初期化に長い時間のかかるコンポーネントや、特殊な実行環境を必要とするポータビリティの低いコンポーネントの場合も、待ち受けサービスを展開してサービスとして共有するのが妥当であろう。

リスナモードでは、UIMA コンポーネントを UIMA-AS のウェブサービスコンポーネントとして自動展開する機能を提供する。Kachako は指定されたリモートサーバにローカルモード同様にサービスを展開する。コンポーネント実行に必要なリソースは、Hadoop の場合と同じように自動解決し転送される。

ウェブサービスではスケーラビリティが常に問題となる。Kachako は ActiveMQ の機能を利用してマスタと複数台のスレーブによる分散並列サービスを自動構成できる。マスタがロードバランサとして負荷分散するよう実装されており、全体がスケーラブルな単一の UIMA-AS サービスであるかのように振る舞う。サービスの利用者は全体を UIMA コンポーネントとして一般のコンポーネントと混合して利用できる。

ユーザの作業はマスタとスレーブのサーバ名を指定することだけである。Kachako のサーバ設定 GUI は Hadoop の構成に加え、リスナモードの構成・サービスの展開中止や負荷の状況監視もサポートするよう実装されている。

4. おわりに

Kachako は誰もが自然言語処理を容易に行えるよう全自動化を提供するシステムで、広汎な技術要素を調和的に統合して実装した。今後は研究協力等で応用例を広げつつ、機械学習など Map/Reduce が必要なコンポーネントの追加と統合、多言語・多モーダルなコンポーネントの追加と対応を進めていきたい。

謝辞

本研究は、JST 戦略的創造研究推進事業さきがけ・文部科学省科学研究費補助金（21500130）の助成をうけて行われた。

参考文献

Cunningham, H., Maynard, D., Bontcheva, K. and Tablan, V. (2002). GATE: A framework and graphical development environment for robust NLP tools and applications. 40th Anniversary Meeting of the Association for Computational Linguistics (pp. 168-175). Philadelphia, USA.

Ishida, T. (2006). Language Grid: An Infrastructure for Intercultural Collaboration. Proceedings of the International Symposium on Applications on Internet. IEEE Computer Society.

狩野芳伸・辻井潤一. (2008). UIMA を基盤とする相互運用性の向上と自動組み合わせ比較 : 国際共同プロジェクト U-Compare. 情報処理学会研究報告. 自然言語処理研究会報告, 2008(67), 37-42.

砂山渡・高間康史・ボレガラダヌシカ・西原陽子・徳永秀和・串間宗夫・松下光範. (2011). テキストデータマイニングのための統合環境 : TETDM プロジェクト. 電子情報通信学会 NLC 研究会, 111(119), 15-20.