Abstract

In this work, we present a workflow management system that allows the modelling, execution and monitoring of workflows oriented to the field of Natural Language Processing (NLP), which is a field that requires large amounts of processing power. This field imposes requirements, which we build our system to answer. These requirements include (1) using a language that is simple to learn, even for non-programmers, but at the same time possesses the expressiveness to enable the modelling of complex workflows, (2) support for grid environments for components that need to process large amounts of data, and the extensibility to allow the addition of modules to work with other environments, (3) error tolerance that allows recovering an error without restarting the workflow and (4) the ability to easily make changes to a workflow.

The workflow management system that this work presents is implemented using a client-server architecture, where, after deploying the server, the users access it using a browser interface. The system allows the specification of workflows through a graphical interface where components can be linked together to create a workflow with little need for coding.

1 Introduction

Natural Language Processing is a branch of artificial intelligence that studies the process of human language understanding and automated generation. The goal is to get computers to perform useful tasks involving human language [13]. This field is similar to e-science fields, such as astronomy and gravitational wave science, where large amounts of data must be processed.

One approach to compensate for this is the use of grid computing, enabling collaborative use of resources to raise system performance. This is essentially a divide-and-conquer approach, where, for example, one large file with 3 Gigabytes can be divided into 6 smaller files with 500 Megabytes each, which are processed by 6 different CPUs.

Another way to boost performance is to optimize the interaction between components. Usually, NLP systems are created from the integration of a large number of different stand-alone components that perform simpler tasks, such as part-of-speech tagging, information extraction, or word sense disambiguation. By optimizing the sequence in which those components are invoked during the execution of an instance of the system, we can further improve the performance of the system; for example, take a system with 3 components (A, B and C) and say that B’s input was the output of A. Normally, we would invoke A, B, then C. However, component C does not need to wait until the other components finish before starting. By invoking A and C simultaneously on different CPUs, the system’s performance can be improved. Furthermore, this approach can be combined with the strategy above by running each or a subset of the components A, B, and C in separate CPUs.

The former generally requires a grid computing environment, where available work is managed and distributed to the available resources. This is generally done using workload management systems such
as Condor [18]. The latter can be achieved by creating and synchronizing multiple processes/threads so that the system progresses accordingly to the dependencies defined between the components within the system. This task is easy when the system is only composed by a small amount of components with few dependencies between them but the complexity escalates rapidly as systems grow larger. As a means to make this process simpler, workflows are often used.

A workflow can be defined as a series of coordinated computational tasks according to a set of procedural rules to deliver a service. A workflow model is the definition of workflow and its instantiation is denominated a case. The actual execution of these cases is handled by a workflow management system. A useful property of workflows is that they do not need to know about the applications that execute the tasks, thus allowing subsequent changes to be incorporated into those applications without compromising the integrity of the workflow. Finally, workflows are a particularly useful way to model systems, providing users a high-abstraction view of the system as a whole and how the system evolves during its execution.

Furthermore, it is convenient to separate the development and the execution of applications into different concerns [5]. At the development-level, the application can be modelled independently of the execution environment as abstract workflows. However, at the execution-level, the individual tasks need to be implemented and assigned an execution environment.

1.1 Problems

Most workflow management systems are aimed at providing services to a general public rather than a specialized one. Because of this, some problems may arise that are specific to the area of NLP.

- Each of the NLP components generally takes years to develop, and for that reason, they are usually implemented by independent research groups, who are more focused in their particular projects rather than on the whole system. As a result, when integrating these components, compatibility issues are bound to emerge. Those components might be working under distinct platforms and be implemented in programming languages, making the programmer go through the unnecessary trouble of writing additional pieces of code to integrate each pair of inter-communicating components. To make matters worse, NLP components are rarely used by one particular project exclusively, but are frequently implemented with a tight coupling to the particular project that they were made for. As a consequence, it becomes harder for them to be reused in other projects. The answer to this problem for most workflow management systems is to provide a heterogeneous service, such as Web Services. However, most developers in NLP are not familiar with these technologies, and do not want to adapt their scripts to allow their invocation by the workflow management system.

- Another problem lies in the fact that, in the field of NLP, as in many other e-science fields, not every researcher is necessarily a programmer. Some of them can be, for instance, from a linguistic background and have little programming skills. In contrast, mainstream workflow solutions on the market, such as BPEL4WS, are oriented for programmers and have a steep learning curve making them not the best choice for developing NLP applications.

- Very few workflow management systems available provide support for grid computing, since high-throughput computing is not required for most industries. However, in NLP, this is considered a requirement, since single components can take hours or days to finish without it.

1.2 Goals

The purpose of this project is to build a workflow management system capable of leveraging the potential of the workflows to provide a more agile development of NLP applications. This will be achieved by accomplishing the following sub-goals:

- Promote the workflow paradigm: the framework must allow the creation of workflows and their execution. Moreover, they must be flexible, allowing different configurations build with little effort.
• Improve the intrinsic-interoperability of components: the NLP components must be able to communicate easily with other components, regardless of the programming language and the platform they are implemented in.

• Support for grid computing: the framework must bridge the workflow management and grid environments.

• Separation the roles within the organization: the system must draw the boundary between the programmers and the software engineers, much like in SOC (service oriented-computing) [6]. While the former are focused in particular components, the latter use those to build a system.

• Provide a graphical interface that is easy to learn and use, for non-programmers and programmers alike.

2 Workflow Management System Model

Our system is based on the model proposed by David Hollingsworth [12], and we will focus on the Application Interface, which dictates what exterior entities can be called during the execution of workflows, the language that will be used to model workflows, and the engine that will execute them.

The application interface will need to include entities that enable grid computing but also offer a large variety of components to use. An example of the former is Condor [18], which is a specialized job and resource management system for compute intensive jobs [18]. , and Web Services [22] are an example for the latter.

The language will need to have high expressiveness, which means that languages that allow complex constructs such as conditional routing, parallel execution and arbitrary loops are preferred. This will be judged based on workflow patterns [1]. This document describes a series 20 patterns through which the expressiveness of each language can be compared. Other important aspects are the usability and analysability of the language and, finally, the range of applications that support it. The possible candidates for the language were PNs (Petri Nets) [20], YAWL (Yet Another Workflow Language) [11] and UML (Unified Modeling Language) Activity Diagrams [16] and BPEL4WS [17]. After analyzing these languages we chose YAWL based on the following conclusions:

• In terms of expressiveness, YAWL is the best choice, since it supports every pattern with the exception of the Implicit Termination, which was done deliberately so that the user needs to think about the finishing conditions of his workflows [11].

• A strong reason for the use Petri Nets is the fact that they are analyzable. PNs and YAWL, which is based on PNs can be analyzed for properties such as soundness and measures including response times, waiting times, and throughput [19]. On the other hand, The analysis of UML activity diagrams has not been explored as much as PNs, and the correctness of BPEL4WS can only be verified at runtime [4].

• The PN-based modeling languages are the simplest to use. Complex behavior such as parallel routing and loops are all modeled in terms of a small set of constructs (places, transitions, arcs, and tokens). Moreover, the language is mathematically formalized, despite having a graphical nature, leaving no room for ambiguous interpretations. Once again, YAWL inherits the same set of advantages as PNs. UML activity diagrams have some similarities with PNs, in the sense that they are graphical. Yet, they are not formalized by OMG[16]. Finally, BPEL4WS has the problem of being language based, making it harder to learn than graph-based languages. The language requires the designer to become familiar with numerous concepts before being able to create processes efficiently. Furthermore, its original design goals say clearly that BPEL4WS is not concerned with graphical representations or a graphical design methodology [14].

• Although BPEL4WS has not been around for as long as PNs, but due to its popularity, there is a large variety of solutions (both proprietary and open-source) for BPEL4WS on the market, which comprise not only of a wide variety
BPEL engines, but also tools for verification of BPEL4WS processes and visual editors.

The engine well need to allow the execution of the language that allows the execution of instances of the chosen language. If the language of the engine is different that language, a conversion will be needed to adapt the specifications to that engine. The engine also needs to allow the definition of the Application Interface defined above. One choice is to use DAGMan [18] which is a service for executing multiple jobs with dependencies in a declarative form and it is well integrated with Condor. However, the DAGMan’s language is a less expressive than the languages analysed above, which is a problem if we wanted to convert a YAWL specification to a DAGMAN script, but also does not allow the invocation of Web Services. The happens with P-grade [15], which is a workflow management system for the development of complex grid applications supporting the design, execution, monitoring and performance visualization. It supports the execution of applications on different environments, such as Condor, Condor-G, Globus, and Dagman. BPEL does provide support for Web Services, and can execute Condor Jobs by using a Web Service interface for Condor, but is still less expressive than YAWL. We could have used the BPEL language instead, but the fact that combines the best expressiveness, analyzability, and usability was a deciding factor for choosing YAWL. The only downside is the fact that it only has one implementation and lacks the support languages like BPEL possess.

3 YAWL

YAWL is mainly composed by its engine, which is responsible for deploying, validating, and executing workflow specifications. After a workflow specification is deployed, instances/cases of that workflow can be launched. A key aspect that distinguishes the YAWL from other workflow management systems is that, while in traditional workflow management systems, the engine is responsible for taking care of “What”, “When”, “How” and “By whom”, the YAWL engine only takes care of “What” and “When” [2]. It does this by keeping a list of available services, called Custom Services, and their respective URLs where the service is deployed in its repository, and then the binding for each node to each service is defined, by the user, within the specification. Having these two elements, when the engine determines that a node is to be executed, the engine fetches the URL of the service to invoke for that node and sends it an HTTP message to the service, announcing that the node is waiting the be executed. The HTTP interface between the engine and the service is defined using a REST architecture [8]. The YAWL distributions include a handful of services, including a service to invoke Web Services, but users are free to create additional services (Custom Services) that can be invoked during the execution of workflows.

The YAWL specification is the document that describes the process to be executed by the engine. By parsing this document, the YAWL engine builds the internal representation of the workflow and deploys it, allowing instances to be launched from that point on. We will be giving an overview of the main elements within a YAWL specification, as it will be needed to understand some aspects of our architecture.

YAWL also comes with the Resourcing Service [10], which is accessible through a web interface. This Service allows the assignment of Work Items within the workflow to human resources within an organizational context.

4 System Architecture

Our system (see figure 1) is composed by a number of loosely coupled components that are connected by either HTTP [21] or RPC [3]. The architecture is composed by 2 web containers, one where we will deploy the YAWL engine and another for our own workflow server. On the YAWL engine’s side, our work will be focused on extending the YAWL’s operational perspective, by implementing additional Custom Services. The following sub-sections describe the each of these components in detail.

The Application Interface is defined by the set of Custom Services that were implemented. These include the Condor Service, which can be used to create and manage Jobs in a Condor environment, and the HTTP Service and Web Service that allow the access to various online resources.

The Workflow Server is used to store and man-
Figure 1: Architecture Overview

age the workflows, users and convert the specifications given by the user to YAWL specifications. This allows the us to extend some functionalities not present in YAWL such as the referencing other workflows that are stored in the server in another workflow. The server can also modify some workflows to add new functionalities such as the monitoring of the nodes within within the workflow, or error handling nodes in case something goes wrong.

The User Interface is the interface that allows that creation of workflows graphically, among others. This interface was developed using the Google Web Toolkit (GWT) [9] and the EXT JS [7], which provide high-level interface elements such as windows, forms, trees and grids. They also provide the abstractions needed to easily implement some functionalities such as sending AJAX messages, which are used to access the server. The interface consists of a desktop where applications can be launched from a toolbar. Of course, the main application is the Workflow Editor used to develop workflows (see figure2) and the Workflow Monitor that monitors the progress of workflows.

5 Features

The following list presents the main features of our workflow management system.

- Workflow Importing - One of the main goals of our system is to allow the reuse of NLP modules. This applies to workflows as well, since they can be viewed as modular tasks and used by other workflows. There are two problems with using other workflows in a workflow. First, the referenced workflow might not be deployed. Second the referenced workflow might be modified, making the workflows that reference it fail. To solve the first of the problems mentioned above, we decided to embed the referenced workflows into the workflow itself. This makes the referenced workflows be deployed with the workflow. The second problem is solved by creating a version system for the workflows. Every time a workflow is saved, a new version of that workflow is created. When a workflow references another, it will actually reference a version of that workflow, which means that the referenced workflow will not be modified. This way, users will not have to guarantee that all the workflows that reference his workflow still work after making alterations. In case the user wants to use the updated version of the workflow, he can change the version of the workflow in the Workflow Node. Furthermore, having a versioning system has the additional advantages of having a versioning control system, which include the ability to go back to a previous version of the workflow. Also, in a situation when two or more users edit the same workflow at the same time, their updates will not be lost.

- Workflow Monitoring - Instead of simply running a workflow and presenting its results, we
wanted to allow the monitoring of the workflow during the execution. From the start to the end of a workflow’s case, our system allows keeps the information regarding the nodes that are executing and the order the nodes were executed, plus other information the user can define.

- **Error Tolerance** - The system allows the definition of error handling functionalities that allow the workflow, in case of errors, to be recoverable. For instance, when a node fails, the user can define a task that prompts a certain user, the next time he logs in to the system, tell the system what to do.

- **Condor Support** - Our system allows the creation and management of jobs in Condor, this is important for running applications that require large amounts of processing power. This is done using the Birdbath which is a Web Service interface for Condor.

6 Evaluation

This chapter discusses the strengths and weaknesses of our system.

The most obvious advantage of our system is that it allows us to build our applications graphically. This has the benefit of providing a more accessible framework for programmers and non-programmers to define complex applications. Moreover, it provides a visual definition of the system that can be presented to the stakeholders of the application. The main problem with the interface that limits part of the usability of the framework is the fact that there are two elements of the interface that require the user to be familiar with the XQuery language that the YAWL engine uses. The first element is the mapping between the variables and the parameters of the components and the second element are the predicates or conditions that control the control flow aspect of the application. However, the interface helps to a certain extent, in that it provides the mechanisms to generate the code for accessing variables and parameters of the application, allowing simple queries such as the direct mapping between the output of a component to a variable achievable without knowing the programming language.

Another advantage of our system is that the YAWL engine that we use to execute the workflows allows various workflows that other workflow en-
gines do not. Some examples include workflows containing cycles and the ability of one branch in the workflow to split into an arbitrary number of branches. There were also some features in YAWL that are not supported in the current version of our system, namely, the creation of multiple instances of a component and the definition of cancelation sets, which were not implemented in our system, since they are currently not needed to test the system.

A more philosophical impact of the system is the fact that it forces the users to separate the roles of software engineer and programmer. The difference between these roles is that the software engineer will use our system to design the whole application/workflow in a highly abstract fashion, and leave the implementation of the individual components of the workflow to different programmers that will use our system to submit those components. The submission of the components can be done in various ways. The advantage of submitting those components into our system is that they can be used in other applications as well, which is useful in the field of NLP since many components, such as part-of-speech taggers, are used by various NLP applications. This allows saving some work in new applications since, as more applications are built in our system, more components that are generally used in this or another field will be registered, and can be used simply by searching the existing components that are registered in the system and inserting the corresponding node in the workflow.

The workflow monitoring feature of our system allows not only the debugging of applications but also their evaluation. For instance, if we want to improve the efficiency of an application we can use the workflow monitor’s information to determine the components that are the bottlenecks of the whole application.

Finally, our system also allows programmers to create error handling mechanisms. An useful example is to allow a Condor Job that goes wrong to be recoverable, instead of having to repeat the whole application that might have been running for hours. By combining different services, we can create interesting error handling mechanisms, such as sending an e-mail and notifying certain users, which can be determined at run-time, that something unexpected has happened and go to a service to receive and parse a reply from that user on what to do next. Although, this is not entirely exclusive to our system, and can be done in most programming language using libraries to make it easier, we still see many applications that do not handle errors since when programming those applications the programmer does not want to waste time to make a error handling module. In our system, we would only have to implement the component that sends an e-mail and parses the reply and sends the required parameters to the output, and all users can then simply insert that component to their application.

7 Conclusion

In this work, we presented a system that allows the implementation and execution of complex NLP applications in a high-level of abstraction without the need to have a deep understanding of programming, using workflows.

The execution of the workflows is delegated to the YAWL engine, which uses the YAWL language that has a high level of expressiveness allowing the definition of complex workflow constructs such as arbitrary cycles and also mechanisms for cancelation and advanced synchronization, thus surpassing the expressiveness of other workflow engines such as BPEL. Furthermore, the YAWL language is based on Petri-Nets, which is simple to process, since a workflow is simply defined as a set of nodes and edges.

The current version of our system allows the definition of workflows in a graphical editor. These workflows are submitted to a workflow server, where they are stored, managed and scheduled for execution. By storing the workflows in a server, all the work that is submitted to the server can be reused in other applications with little effort. Additionally, the system also keeps track of the various versions of the workflows to allow users to freely update their workflows without worrying about making other projects that reference it unusable.

The communication between the engine and external applications is done through a group of loosely coupled services called YAWL Services. Through these services, the engine is capable of communicating with multiple environments, including Web Services, HTTP, and Condor. The system
also allows the implementation of additional services, which can be registered with the system without having to recompile or restart it.

Finally, the system also provides workflow monitoring mechanisms that allow the state of the workflows to be known. This is done through an event-based mechanism, where whenever a node starts and ends, a event registered. Furthermore, the user can define failover behaviors for his components allowing them to be recovered without restarting the workflow.

References


