**University of Alberta**

# A LIGHTWEIGHT COORDINATION APPROACH FOR RESOURCE-CENTRIC COLLABORATIONS

by

**Morteza Ghandehari**

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

**Master of Science**

Department of Computing Science

*To my family*
*For their unconditional support*

# Abstract

A very common form of collaborative work involves people working on shared resources, such as, for example, co-producing a project report, including editing text, cross-referencing citations and validating the budget or reviewing and authorizing different aspects of a loan application. All these types of collaborative work share a few key properties. They usually involve a variety of interactive tools. The various process steps are not precisely ordered, but have some logical inter-dependencies among them. Although some steps may be automated, the process is driven primarily through interactive tools by people, who need to notify each other. Finally, these processes usually take a long time to complete.

Web-based collaboration is the norm nowadays. Collaborative editors, such as wikis for example, coordinate people working on documents, but are neither powerful enough to support coordination more complex than notifications, nor amenable to integration with other tools. On the other end of the spectrum, classic business-process management systems are powerful enough to cover complex coordination requirements, but are too complicated to use and too regimented in the types of coordination they support. As information is becoming increasingly available and shared through REST APIs, there is a need for enabling web-based collaborative systems to support resource-centric collaborations.

To meet the need for flexibly coordinating people, interactive tools, and automated services, we have developed a coordination approach and a supporting framework. Our solution consists of (a) a language and tool support for specifying collaborative activities and the resources they manipulate, (b) an engine for enacting them at run time, and (c) a systematic methodology for integrating the engine with the various interactive systems and services involved. Our framework balances expressiveness and simplicity and its usefulness has been demonstrated in two real-world projects.

# Acknowledgements

I would like to give her my sincere thanks my supervisor, Professor Eleni Stroulia, for her invaluable guidance, support and constructive feedbacks. I would like to express my utmost gratitude to her because not only she is an absolutely wonderful supervisor but also her personality is and always will be an inspiration for my life.

I would also like to thank my colleagues in GRAND and CWRC projects who helped me a lot in this thesis.

Finally, I extend my special thanks to Parisa Delfani whose support and encouragement was always with me.

# Table of Contents

# List of Tables

# List of Figures

# List of Listings

# Chapter 1

# Introduction

Today we are witnessing an abundance of technologies, conceived to support the development of web-based applications, in general, and web-based service-oriented systems, in particular. In the service-oriented paradigm, processes are typically supported through service orchestration; the various steps of the process are delegated to invoked services, and service orchestration is employed for coordinating the operations. Traditionally, the steps are implemented by WSDL/SOAP web-services operations, and the process model is specified using WS-BPEL as the de-facto standard language for web-service orchestration; furthermore, the process itself can be published as a new service, implemented on a WS-BPEL engine.

More recently, the REST (Representational State Transfer) [23] architectural style has emerged as an alternative approach for development of web-based systems and there are three reasons for this phenomenon.

1. First, the REST approach is conceptually and syntactically simple; it relies on the HTTP protocol, with XML (or JSON) as the exchange format for the payload data, and a simple syntactic style for formulating HTTP requests as traversals of the XML schemas of the underlying resources.

2. This simplicity makes the related learning curve smoother, since developers can easily migrate from standard web-based development to exposing their systems as resources accessed through REST APIs.

3. The increased availability of interesting information and simple development tools (some based on the demonstrational-programming paradigm) through REST APIs has spurred the interest of web users to develop information mash-ups.

Given the conceptual simplicity of the REST style, many service providers are now publishing their services as REST services, instead of, or in addition to, WSDL/SOAP web-services. For example, Google stopped supporting its SOAP Search API, after it introduced its REST Search API [4]. At the same time, governments, following the "open data" movement, are making rich information repositories publicly available. As a result, the number of information resources accessible through REST APIs is increasing and so does the number of web-based interactive tools for using and manipulating these resources. The question then becomes how to coordinate the activities supported by these various tools.

In this work, we focus on resource-centric collaboration workflows. The term *resource-centric collaboration* refers to a type of human-intensive workflows, in which a group of people work together on shared information resources. The information is available to the collaborators in a resource-centric environment and can have various types such as XML data, text files, or bibliographic references. The objective of the collaborative activity is to develop and manage the various resources by taking turns editing them, annotating them with meta-data, and evaluating their degree of progress and completion. Examples of this type of collaborative work include co-authoring a scholarly publication, producing a project report, and coding/annotating an original text with meta-data.

Let us consider a simple project-report coauthoring collaboration, where a group of students work together to produce a report on their team project. A team member may develop the first version of the report and provide it as a resource on the web. Then other team members may take turns editing the report, with the turn-taking order being ad-hoc or possibly depending on the members' roles in the team. Each team member, when she is done with her turn, indicates the availability of the resource to the team, possibly through an explicit notification mechanism. At some point, a team member, who has been designated to regularly inspect and assess the quality of the report, may decide that the report is no longer a "draft" but has been "completed" and submit the report to their supervisor for approval. The supervisor receives the report and evaluates it. If the supervisor approves it, the resource is "published"; otherwise, the report is sent back to the team, possibly with specific comments to be addressed, for further editing.

In general, resource-centric collaborations are asynchronous collaborative activities [19] which mainly involve accessing and manipulating shared information resources. Various operations of the collaborations are performed by different users on different resources at different time using different tools. In order to coordinate the operations, some supporting processes are also engaged including informing users about the operations and status of resources, providing the users with the capability to make decision in a collaborative manner, invoking automated operations on resources as necessary, and enforcing the dependencies among resources and operations.

Resource-centric collaborations, of which the above project-report coauthoring story is an instance, share several common properties.

- **They involve many "people" activities**: While various editors and tools are involved in the overall activity, the collaborative activity is initiated and driven mostly by the people participating. In the above mentioned example, it is the responsibility of each team member (a) to receive the URL of the resource and start editing, when they are notified by the system, and

(b) to decide when to release the resource to the rest of the team. Contrast this with automated workflows where the various steps are performed by automatically invoked software services that explicitly signal their completion with their return parameters.

- **Their steps are loosely ordered**: Resource-centric collaborations are semi-structured. In other words, the control over the various tasks is non-deterministic, and in many cases, a particular task can be performed any time. In our example, although the report may consist of specific sections, the order in which these sections are edited is unlikely to be fixed; any section can be edited any time and it is only important that all the sections are written before the report can be considered "completed". In contrast, in automated workflows, although there may be alternative control-flow paths, each one is annotated by an explicit condition on when it is taken.

- **They have simple structure**: The process models of resource-centric collaborations usually do not have complex control elements, computation, or data transformations. They usually involve the evaluation of conditions (e.g. to assess whether a particular person may access the resource), simple service calls (e.g. to notify the persons involved), and value assignment (e.g. to manage the transition of the resource through phases). In contrast, web-service orchestrations must support the mapping of complex parameters through services, the maintenance of global variables, and the evaluation of complex control structures. Therefore, languages such as BPEL[43] are too complex and a simpler language would be more appropriate for specifying them.

- **They have undemanding performance requirements:** Finally, resource-centric collaborations do not usually have complex or strict performance requirements. For example, automated workflows may indicate upper limits in the response time of a service (and upon its expiry the middleware may invoke a fall-back alternative). In contrast, resource-centric collaborations may need to conform to deadlines for the overall completion of the coordinated task, but it is unlikely that any of the individual activities involved is time-critical.

Current solutions in support of resource-centric collaboration (Figure 1.1) include tools such as collaborative real-time editors. These tools usually provide some basic coordination supports like sending notifications or controlling access. They are relatively inexpensive and light-weight. They also provide the flexibility in specification and ad-hoc modification required by semi-structured nature of resource-centric collaborations. However, they are not powerful enough to support customized or more complex coordination requirements, such as monitoring deadlines, or ordering of steps, and they cannot be easily integrated with other tools [41]. On the other extreme in the coordination spectrum, classic business process management systems, e.g. BPEL systems, are powerful and capable of supporting various coordination forms. However, they are costly and complex to operate and maintain. In fact, these systems are too heavy-weight based on the requirements of

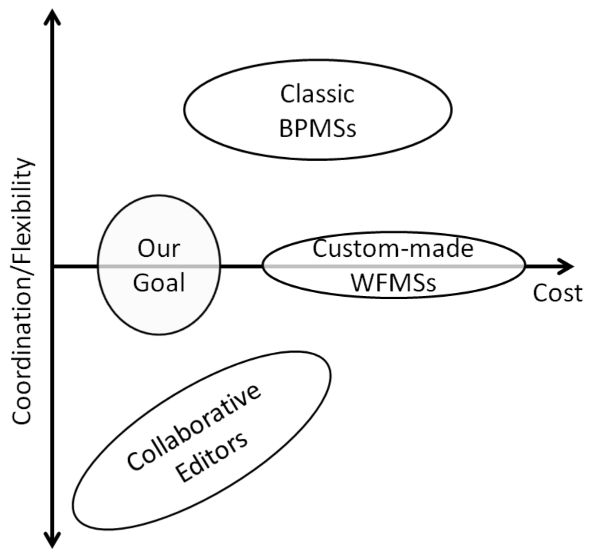

Figure 1.1: Comparison of different solutions for supporting resource-centric collaborations

resource-centric collaborations. Another solution is to develop a workflow management system from scratch. While a custom-made system can be tailored according to the requirements of the project, developing a comprehensive workflow management system is far from trivial and substantially increases the effort required for developing the collaborative system.

Given the relative abundance of resource-centric collaborative activities, the problem of developing a systematic method and a tool framework for supporting resource-centric collaborations by integrating the interactive tools that manipulate these resources is compelling. In other words, the problem is to add the coordination support to a multi-user, multi-tool resource-processing environment on the web. In this thesis, we describe exactly such a collaborating system that we have developed for supporting resource-centric collaborations, which properly balances the required flexibility and coordination.

The main contributions of our work include:

- We introduce an approach for coordinating people, tools, services, and resources in a service-oriented resource-centric environment. The approach enables the collaboration of users on shared resources, with the goal of creating and maintaining these resources. This approach utilizes many principles of REST style on different levels such as uniform interface and stateless communication. It orchestrates a number of REST services in order to publish collaborations as new REST services. More importantly, the approach itself is designed according to the philosophy and properties of REST style since simplicity, flexibility, ease-of-use, and agility were the main design goals.

- We designed a language for specifying collaborations. Using this language, the collaboration specifications define the required coordination among people, tools, services, and resources.

As opposed to web-service orchestration languages that are XML-based, we employed a scripting paradigm for our collaboration language; we also tried to make the language intuitive enough so that it can be used by non-technical users.

- We developed a comprehensive software framework for supporting resource-centric collaboration, composed of (a) a collaboration editor for writing the collaboration specifications in the above language; (b) a compiler for translating the collaboration specifications into executable collaborations; (c) an engine for executing the collaborations at run time; and (d) a set of tools for managing the service and integrating it with the systems that the collaborators may use to manipulate their resources. The system is powerful enough to support the coordination required by resource-centric collaborations; at the same time, it is light-weight enough to be used in resource-centric environment such as web-based systems.

- We designed a methodology for integrating the collaboration system within an ecosystem of other systems and tools. It is in fact an approach which aims at facilitating the integration process. We also provide two case studies which demonstrate our collaboration system in action. These case studies are evidences of applicability of our collaboration system in real-world projects and usefulness of our integration model.

The remaining of thesis is organized as follows:

**Chapter 2 (Related Work)** describes the background for this thesis and the related work. We start by explaining some of the fundamental concepts used through this thesis. Then, we explore the research areas relevant to the problem tackled in this work. Furthermore, some related work focused on the similar problems is introduced and compared to our work.

**Chapter 3 (Collaboration Language)** defines the collaboration language we designed for specifying collaborations to be supported by our collaboration system. We first describe the basics of the collaboration language including the motivation behind designing a new collaboration language, the paradigm of the collaboration language, and its elements. In order to familiarize with our collaboration language, a sample collaboration specification written in the language is demonstrated early in this chapter. In addition, we provide the complete specification of the language consists of the lexical elements, the syntax, and the semantics of the language.

**Chapter 4 (System Architecture)** describes the architecture of the collaboration system. In this chapter, the components of the collaboration systems and the dependencies among them are described in details. For each component, we outline the main responsibilities of the component and explain its structure and behaviors. In addition, some noteworthy details about the implementation of the components are also mentioned through the chapter. Finally, we explain how the components interact in order to execute collaborations.

**Chapter 5 (Integration Model)** describes our suggested model for integrating the collaboration system with other systems. In this chapter, we describe the components of this model, i.e. the components which should be developed for enabling the integration of the collaboration system with other systems, and their interactions.

**Chapter 6 (Case Studies)** presents two projects, as case studies, in which our collaboration system has been employed as case studies. Case studies are about two web-based system projects, named GRAND and CWRC. For each case study, we first explore the system to which we want to add the collaboration support using our system. Then, we adopt our integration model according to that particular project. Finally, we describe how the integration process has been done. In the GRAND case study, we also provide a sample GRAND collaboration and its corresponding collaboration specification written in our collaboration language.

**Chapter 7 (Conclusion)** starts with a summary of the thesis. Then, we discuss about this work and its contributions. Finally, we suggest some possible future work based on this thesis.

# Chapter 2

# Related Work

This thesis focuses on providing support for the coordination of people, tools, and services in order to enable interaction-intensive collaboration of people on shared web-accessible resources. According to this description, our work relates to two major areas of software engineering: business process management [30, 46] and computer-supported cooperative work [25]. Furthermore, as our approach is designed specifically for integrating tools accessible through REST APIs, it relies on concepts and practices from the service-orientated software-engineering research area [33, 22].

A workflow, usually considered equivalent to a business process, is a sequence of related activities that produces a result with observable value for business actors [44]. Businesses define and follow their own set of workflows in order to achieve their organizational goals. A *Workflow Management System (WFMS)* is a system mainly used for supporting workflows in organizations. According to the Workflow Management Coalition [27], “a workflow management system is a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of workflow logic”. Instead of focusing on individual activities, WFMSs focus on the way different activities are related to each other, in order to structure workflows. A WFMS consists of two main components: the specification module and the execution module [21]. The specification module enables the specification of workflows, known as *workflow schemas*, and the execution module is responsible for actual enactment of workflows through *workflow instances*. In fact, workflow schemas represent the structure and behavior of workflows; and workflow instances are the instances of workflow schemas at run-time. As we mentioned in the Introduction, resource-centric collaboration is a type of human-intensive workflows; therefore, the resource-centric collaborations are a subset of workflows and the workflow concepts are also relevant to the collaborations. Throughout this thesis, the term “collaboration specification” is in fact mapped to the concept of workflow schema and the term “collaboration instance” is equivalent for the concept of workflow instance.

One of the key properties of resource-centric collaborations is the flexibility required in modeling and modifying the rules of the collaborative activities. Workflow flexibility is one of the major research topics in workflow management and has been studied for more than a decade [26]. There is substantial research on various aspects of workflow flexibility such as ad-hoc modification, flexible modeling, semi-structured workflows, and workflow adaptation. Burkhart and Loos [15], Schonenberg *et. al* [38], and Carlsen *et. al* [17] have surveyed different approaches for enhancing workflow flexibility and evaluated the support of workflow flexibility in a selection of workflow management systems. These approaches address the flexibility requirements of many types of workflows, but they do not provide a specific and comprehensive solution for resource-centric collaborations as they do not intend to. However, many parts of this work were inspired by the researches on workflow flexibility.

The artifact-centric approach [29] to business-process modeling shares some similarities with our envisioned resource-centric collaborations. This approach focuses on the business data and the named artifacts that are manipulated and augmented during the life-cycles of the business workflows. Artifact-centric workflows are centered around the artifacts [32]. There are various studies on artifact-centric workflows, *e.g.* [32, 14, 24], but all of them share the idea of managing artifacts, that capture business goals, and developing services which manipulate artifacts according to business rules. Resource-centric collaboration can be considered artifact-centric, since the coordination is gathered around central artifacts which are resources in resources-centric collaborations. However, the artifact-centric approaches are concerned with the "structuring" of the artifacts, while our approach is agnostic of the internal structure of the resources, which may be the concern of specific interactive tools. Instead, our approach sees a resource as a whole which can be accessed and manipulated through a REST interface. In addition, artifact-centric workflows require pre-defined structural models of artifacts, while such a model can not be provided for resource-centric collaborations as the structures of documents may not be well defined. Therefore, artifact-centric approaches do not provide the flexibility required by resource-centric collaborations.

Another field which is closely related to our resource-centric collaboration approach is web-service orchestration. Web-service orchestration refers to the process of creating composite web-services by combining and coordinating a set of simpler web-services [35]. Orchestration typically delivers executable services that can interact with internal and external web services at the message level. The orchestration engines are used for publishing composite services and executing them at run-time. The *Business Process Execution Language*, also known as BPEL, BPEL4WS or WS-BPEL, is the most popular web-service orchestration language. In fact, BPEL is emerging as the de-facto standard language for specifying inter and intra business processes via web-services [16]. It was originated from Microsoft's XLANG and IBM's Web-Service Flow Language (WSFL). It was first submitted to OASIS consortium for standardization by several software technology corporations including Microsoft, IBM, and SAP. BPEL is a XML-based language used for defining the control logic required to coordinate web-services participating in a workflow [35]. It is defined on the basis of several other web-service XML-based specifications; among them, the Web-Service Definition Language (WSDL) is the most influential one and is used for describing web-services. A BPEL process specification mainly contains the following elements: (1) declaration of the web-services to be orchestrated, (2) description of the control flow among web-service invocations, (3) declaration of the variables used to maintain the state of a process, and (4) declaration of the data flow [45]. BPEL employs various concepts and constructs form scripting and programming languages including scope, assignment, sequential execution, conditional branching, and repetitive activities. In addition, BPEL supports both synchronous and asynchronous style of communication. It also provides some advanced mechanisms such as exception handling, compensation, event handling, message correlation, and parallel processing [20]. In recent years, many proprietary and open-source BPEL engines have been developed to support fully executable BPEL definitions such as Oracle BPEL Process Manager [8] and IBM WebSphere Process Server [6].

BPEL is a powerful language capable of supporting various types of business processes in service-oriented paradigm. However, there are some issues about employing BPEL for supporting resource-centric collaboration in web-base environment since the requirements of resource-centric collaborations are not inline with the design goals of BPEL. In fact, BPEL was formed as the result of joint work by some heavyweights in software industries in order to merge the best practices in service orchestration and develop a standard service orchestration language. Therefore, quality of service and completeness of the language were among the main design goals. As a result, the language is particularly suitable for enterprise settings and B2B interactions; however, the complexity and overhead of the language is in contrast with the flexibility and simplicity required by resource-centric collaborations [18]. In addition, although BPEL is well-suited for automated business processes, it has fundamental weaknesses in supporting human-driven workflows since it does not have any native support for user interactions required in resource-centric collaborations. [31].

Human-driven workflows are business processes that involve people. On the other hand, there are automated business processes which do not require human participation during the execution [42]. Automated processes are usually predictable, relatively static, and independent of human intervention. The data used in automated processes tends to be well structured, so it can be manipulated by applications and web-services. Human-driven workflows may contain both human participants and automated activities. These workflows are more dynamic and flexible. The data used in these workflow are usually less-structured and human readable. Besides, human-driven workflows usually have reactive and event-driven nature. In recent years, some solutions have been proposed which aim at enhancing BPEL in supporting human-driven workflows [28]. BPEL4People is the most prominent one among these solutions. BPEL4People introduces a extension for BPEL to address human interactions as a first-class citizen. In fact, BPEL4People enables BPEL to integrate role-based human activities in orchestrations. First, IBM and SAP published a white paper, named BPEL4People, in which the importance of human-driven workflows is emphasized and the requirements of enabling BPEL to support this type of workflows are identified [31]. This white paper describes scenarios that involve human interactions and cannot be specified with BPEL, and motivates and outlines appropriate extensions to BPEL to address these scenarios. People activities, human tasks, and people links are the most important elements introduced in the white paper. *People activity* is a new type of activity which integrates human interaction within a process; *human task* defines the actions that a human participant must perform; and *people link* is used to associate people activities to human participants. In addition, BPEL4People introduced constructs for working with user information such as roles, users, and groups of users. Later on, a group of several heavyweight software corporations published two specifications [12, 11], namely WS-Human Task and WS-BPEL Extension for People. These two specifications use the ideas outlined in the white paper and together provide a concrete realization for them. The extensions are designed in a way that they are layers over the traditional BPEL web-service stack and enhance basic BPEL services to support human workflows.

Since REST services [23] emerged as alternatives for web-services, it has become important to support REST services in orchestration languages such as BPEL. In addition, as REST services are simpler and more flexible than web-services, incorporating REST services in BPEL process will cause improvement in flexibility and simplicity of the whole BPEL process; therefore, it would be more aligned with the requirements of resource-centric collaborations. As a solution to introduce REST style in BPEL, Pautasso [34] has proposed an extension for BPEL, named "BPEL for REST", which aims at enabling BPEL to support the native composition of REST services in addition to WSDL-based web-services. This extension is mainly composed of two parts: (1) enabling BPEL to directly invoke REST services; (2) publishing the state of the process as a resource which can be accessed through REST service. For invoking REST services, four new types of activities were introduced, namely "GET", "POST", "PUT", and "DELETE". As it is apparent from their names, these activities are used to invoke the corresponding HTTP methods on the given resources indicated by their URIs. For publishing processes as REST services, the "resource" container element was introduced. This element creates and publishes a resource to the clients. The element may contain a set of request handlers. Every request handler has a type which is one of the four defined types: "onGet", "onPost", "onPut", and "onDelete". A request handler is executed whenever the corresponding HTTP method is called on the enclosing element.

The extensions for enabling BPEL to have native support for human interactions and REST services have made BPEL more suitable for supporting resource-centric collaborations. Still, BPEL cannot be considered as a genuine solution for supporting resource-centric collaboration as the design philosophy of BPEL is not inline with the flexibility and simplicity required by resource-centric collaborations. In fact, BPEL is overly complicated and verbose for these types of interactions. Furthermore, BPEL follows an activity-based process model which is better suitable for automated activity, while resource-centric collaborations are more event-driven nature; essentially, user-interaction events within one tool may signal notifications to other users who may use yet other tools to continue the collaboration. In our approach, we deliberately opted for simplicity and flexibility instead of completeness and quality of service. Therefore, we believe that our approach is more appropriate for supporting resource-centric collaborations.

Mash-up is a data-driven approach for service composition [13]. Mash-up is used for developing new web applications combining a number of distributed resources providing data, business logic, and user-interface [36]. Mash-up is mainly intended to be used by less-technical end-users; as a result, simplicity and ease-of-use are important issues in designing mash-up systems. Mash-up systems take the responsibility of creating the user-interface for the composed service; in fact, the integration of various resources is done in user-interface level. Typically, the mash-up systems provide graphical or scripting environments to end-users for defining the mash-up, e.g. Yahoo Pipes [10].

Similar to mash-up, our approach proposes a service composition model which emphasizes on simplicity and ease-of-use. However, there are some fundamental differences. First, mash-up is data-driven and focuses on data aggregation; while, our approach is process-oriented and focuses on behavioral aggregation. Second, mash-up targets end-user web-application development; while, our approach is used for workflow-aware orchestration of a set of applications and services. Finally, mash-up approaches are typically involved in creating user-interfaces for the composed services; while, our approach does not engage in this matter.

Finally, we should mention that there are two groups of papers which are focused on the very similar problems to ours. In fact, they aim to address the problem of supporting collaborative workflows in service-oriented paradigm.

Bite [37, 18]is a process-oriented composition language for Web. Bite is in fact a simplified workflow language designed according to REST architectural principles. It aims to enable mainstream web-applications (in general, resource-centric environments) to benefit from service composition. In order to develop workflows, Bite combines various resources including REST services, simple human interactions such as email exchange, collaboration services provided by online collaboration systems, and back-end services such as local java functions; then, it exposes the developed workflows as REST services. In Bite, workflows are published as live resources. The creation of and interaction with workflows is done through HTTP methods, i.e. GET, POST, PUT, and DELETE. More specifically, Bite employs ATOM protocol for managing workflows and workflow instances. Bite uses a lightweight process model, in the form of a flat graph. For simplicity, it does not support scopes, compensation, or transactional support. The model is composed of activities which define units of work and links which defines dependencies among activities. The Bite language elements mainly consist of basic HTTP communication primitives for receiving and replying to HTTP requests; utility activities for waiting, calling local code, or terminating the flow; and controls, such as loops. Bite originally supports only a basic set of activity types and provides two extension mechanisms: (a) the activity extension, which enables defining new activity types in a first-class manner, and (b) the script extension, which allows execution of scripts written in any of the supports scripting languages. Bite adopts many concepts form scripting languages such as dynamic data types or “convention over configuration”. For integrating human interaction, Bite supports emails and browser-based interactions from the workflows. It generates emails and HTML replies using a rendering mechanism and provides them to human participants. In this way, human participants get informed and also can perform actions (by clicking on links or submitting the forms embedded into email and HTML replies). The authors of these papers argue that the proposed language works properly for both data-driven workflow and collaborative workflows. In addition, the authors of the Bite language propose a run-time architecture for executing the workflows specified in the language.

Bite is one of the pioneers in the field of process-oriented service composition for web. It also can be considered as genuine system for supporting collaborative workflows as it addresses some of the main requirements of these workflows such as web-human integration, light-weight process model, and flexible configuration. In fact, Bite’s approach toward supporting collaborative workflow was a guideline for us in designing of our system. However, we tried to go beyond it and design a system which better suits the requirements of resource-centric collaborations. Bite is practically a simplified version of BPEL adopted for web environment; as a result, a sequential activity-based process model which is not appropriate for modeling semi-structured workflows with loosely-ordered steps. While Bite is more powerful in some aspects, we focused on ease-of-use of our language by employing a scripting-style paradigm. In addition, our language supports Roles and Relations as first-class language elements; as a result, our language would be more natural for specifying collaborations which involve context-data manipulation. Furthermore, we provided a more complete solution including an integration model and a complete collaboration management system.

Schuster et. al [47, 40, 39] focused on satisfying the flexibility and coordination requirements of “creative document collaboration”. They define creative document collaboration as a type of collaborative workflows, which involve ad-hoc human interaction and unexpected changes in order to develop evolving documents. This definition is similar to our definition of resource-centric collaboration. For supporting creative document collaborations, the authors propose a service-oriented approach, which consists of a collaboration model and infrastructure architecture. The collaboration model defines the various coordination requirements and the infrastructure architecture is responsible for enacting the coordination. As the first step of the modeling process, a document is modeled as a hierarchy of document pieces, the contributions, which deliver the pieces and the services that implement the contributions. This hierarchy can be refined over time. These services are invoked either manually by a human coordinator, or automatically by a coordination engine. The coordination engine follows rules defined for the collaboration, and accordingly invokes services. Rules are defined in the forms of event patterns and corresponding actions. These rules can be used for defining temporal dependencies such as prerequisite relation or deadline, automating service call, and detecting potential inconsistency.

Many of the requirements identified in this work are similar to the requirements of resource-centric collaborations. In addition, the proposed approach for supporting the collaborations is appropriate for web-based systems. They also employed REST style for designing their infrastructure architecture. In spite of all these similarities, there are fundamental differences between this work and ours. One of the main differences is that their approach is focused only on documents with compound internal structures, while our approach is conceived more generally, to support any kind of resources. Consequently, their approach is specifically involved with modeling and managing the structures of documents, while we do not engage in these matters and see resources as atomic objects which can be modified through REST interfaces. More importantly, we believe that our coordination approach is more powerful than the rule-based model introduced in these papers. The set of events by which the rules can be defined are limited. In addition, the employed event processing language is not particularly appropriate for defining the coordination requirements; therefore, it can be difficult or event impossible to define more complex coordination requirements. Furthermore, our language is more intuitive to be used by non-technical users. Finally, our supporting system and integration model are more complete than the infrastructure architecture proposed by the authors.

# Chapter 3

# The Collaboration-Specification Language

The first step in supporting collaborations is modeling them. A collaboration model, a.k.a. collaboration specification, must define the elementary tasks that the collaborators may perform and the control structures through which these tasks will be coordinated.

We studied various approaches and languages for modeling workflows. As we discussed in the “Related Work” section, there are numerous powerful workflow languages, which are not very appropriate for modeling resource-centric collaborations. In particular, most of the workflow languages are designed for structured workflows and assume a deterministic control over the process steps; therefore, they do not meet the flexibility requirement of resource-centric collaborations. On the other hand, there are some workflow languages which consider flexibility, but they do not provide sufficient coordination for these collaborations. In this thesis, we designed a new language, for modeling resource-centric collaborations. Our “collaboration language” (as we will refer to it throughout this thesis) balances the required flexibility and coordination support.

The language is based on the event-driven paradigm, since resource-centric collaborations are human-driven and, as a result, the core coordination mechanism involves “reacting” to people’s actions. Therefore, a collaboration specification is composed of ECA (Event-Condition-Action) rules. In these ECA rules, “events” are messages generated by external components (i.e., the systems used by the collaborating team members to perform their tasks) or other collaboration instances; “conditions” are logical expressions regarding the state of the collaboration instances or incoming events; and “actions” define behavior as the response to an expected event, under some conditions. The model of our collaboration languages is illustrated in Figure 3.1.

The language supports the following main constructs.

- **Types** are data elements used for storing values in collaboration instances. They may be associated with methods for accessing and manipulating their values. The basic types supported in our language are **Boolean**, **Integer**, **String**, and **Time**. We also defined a special type named **User** in order to provide a first-level support for working with the data of the people involved.

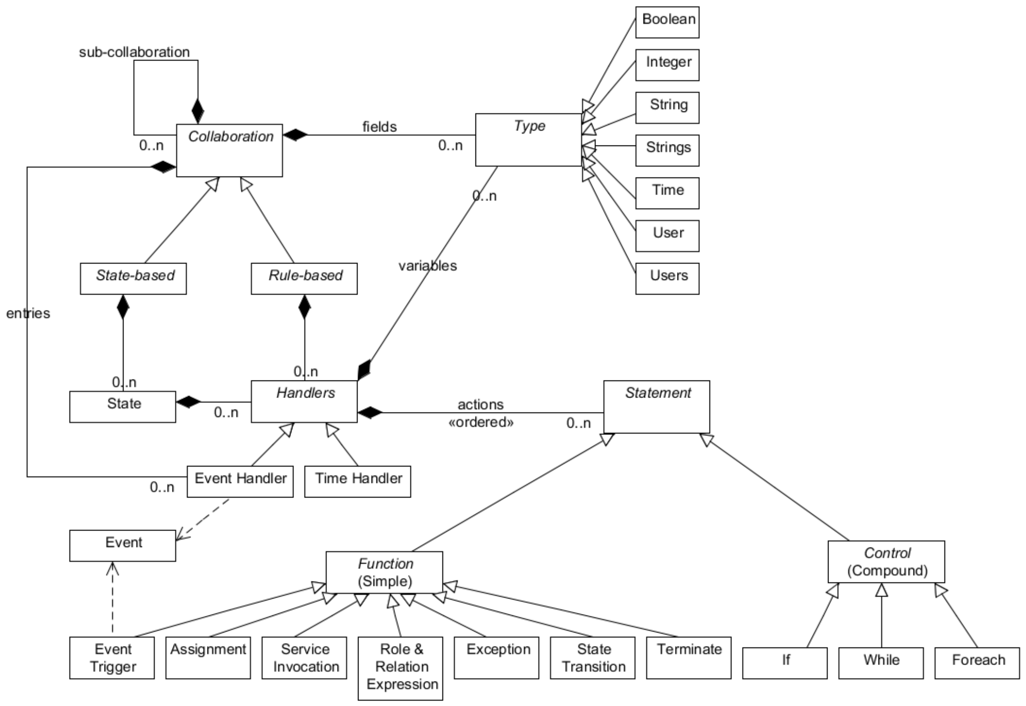

Figure 3.1: The model of our collaborating language

In addition, we included two collection types in the language, **Strings** and **Users**.

- **Functions** are system-level methods used for managing collaborations instances and facilitating interactions. The supported functions include **methods** for invoking web services, **event** triggering, and publishing an **error**. Using these functions, a collaboration instance can communicate with other collaboration instances and external components.

- **Control structures** are used for defining the control flow of the collaboration tasks, based on events and conditions. The language supports the three control constructs of structured programming: **sequence**, **selection**, and **repetition**. Sequencing is achieved through ordered execution of statements. Selection is supported by **IfElse** statement and Repetition can be performed using **While** and **Foreach** statements.

In our language, we designed two styles for specifying collaborations: state-based style and rule-based style. Each collaboration should be writen in one of these sytles. In the state-based style, one has to identify the states through which the collaboration instance passes during its life-cycle and group the ECA rules based on the states to which they belong. In this style, ECA rules enable the transition of the collaboration instance from one state to another. In the rule-based style, there is no explicit concept of state or state transition; thus, a rule-based collaboration specification is roughly a set of ECA rules. Although these two styles support different means for arranging ECA rules, the expressive power of the two styles are equivalent, and a specification written in one style can be transformed into the other. However, there is a significant difference in their applications. The state-based style best fits the collaborations which require more structured models as this style demands complete state-based behaviors of the collaborations. On the other hand, the rule-based style is more natural for semi-structured collaborations as the style allows more flexibility in specifying the collaborations. Finally, it should be mentioned that one system can have collaboration specifications written in both of the styles, and the collaborations can interact with each other no matter in which styles they are written.

In addition to collaboration specifications, a configuration specification should be provided to the system. In this configuration specification, the elements used for interacting with the collaborations are defined. These elements include event definitions, pre-defined service calls, and methods for manipulating roles and relations. In a system, the configuration specification is shared among all collaborations.

In the remainder of this chapter we provide more details about our collaboration language. We first present some simple examples pointing out the applicability of our collaboration language and its capabilities in declaring collaborations. Having a big picture of how the language benefits us in collaboration definition, we then provide lexical, syntactic and semantic details of our collaboration programming language. Accordingly, after reading this chapter the user of the system should be able to define her target collaborations in a format that is understandable by the system.

## 3.1 Two Illustrative Examples

In this section, we present some examples of collaborative processes, specified in our language. Our goal is to show how a collaboration can be described in our system, and how the proposed collaboration language is capable of covering different types of collaborations, i.e., state-based and rule-based. To achieve that, we use the project report co-authoring collaboration described previously in Chapter 1. The statechart diagram of this collaboration is depicted in Figure 3.2.

According to the diagram, the collaboration is initially in the *Draft*ing state, in which students collaboratively edit and update the report. Upon submission of the report, the state changes to *Pending*, where the participants wait until the supervisor of the project decides about the quality and completion of the report. If the report is accepted, the collaboration proceeds to the *Published* state, which is the final state. Otherwise, the collaboration goes back to *Draft*ing, so that the students may continue working on it.

In order to specify this collaboration example, we first need to write the configuration specification for the participating interactive systems. As we mentioned before, we need to define the events, roles, relations and services for interacting with the collaboration. In particular, events are incoming channels for the collaboration by which external components send messages to collaborations; while roles, relations, and custom services are the means by which the collaboration responds to events or receives additional information. A sample configuration specification for this system is shown

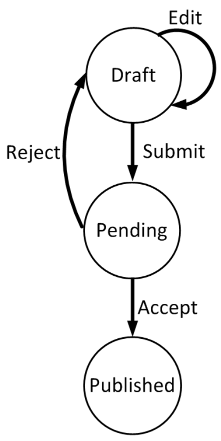

Figure 3.2: The statechart diagram of the sample project report co-authoring collaboration

below.

```
// Event Definitions
Event Create (String projectID, String reportID);
Event Edit ();
Event Submit ();
Event Accept ();
Event Reject();

// Role Definitions
Role Student (uid) : "...url..." , "...url...";
Role Professor (uid) : "...url..." , "...url...";

// Relation Definitions
Relation Supervise(User supervisor, String projectID) :"...url...", "...url...";
Relation Member(User user, String projectID) :"...url...", "...url...";

// Service Definitions
String POST Lock (String reportID) : "...url...";
String POST Unlock (String reportID) : "...url...";
String POST Email (Users receivers, String content) : "...url...";
String POST Publish (String reportID) : "...url...";
```

Listing 3.1: The configuration specification for the sample project report co-authoring collaboration

According to the configuration specification, there are five events defined in the system. The Create event instantiates a project report co-authoring collaboration. The Create event has two parameters: projectID, which identifies the project for which the collaboration is instantiated; and reportID, which identifies the report on which the team works. The other four events cause the state transition as indicated in the statechart diagram of the collaboration.

Roles characterize the roles of the users in the context of the collaborative activity, i.e., Student and Professor in this case. In addition, there are two relations defined in the configuration: Supervise and Member. The Supervise relation is used to check if a user is the supervisor of the project, and the Member relation is used to check if a user is a member of the project. We also defined four services in the system: (a) the Lock method is used for freezing a report, so that nobody can edit it any more; (b) the Unlock method is used for canceling the effect of the invocation of the Lock service; (c) Email is responsible for sending emails to users; and (d) Publish is used to finalize the report and make it public. The details regarding declaration of these elements will be provided later in this section.

Based on this configuration specification, we can now define our collaboration specifications. The specification for our sample project report co-authoring collaboration is provided below.

```
Collaboration StateBased ReportingCollaboration {
  // Field Declarations
  String projectID;
  String reportID;
  Users team;
  User supervisor;

  // Entry Specifications
  Entry Create {
    projectID = e.projectID;
    reportID = e.reportID;
    team = Find ( ? Member projectID);
    supervisor = Find ( ? Supervise projecID);
    To(Draft);
  }

  // State Specifications
  State Draft {
    @Edit [Student]{
      If( ! (team Contains e.Sender) )  {
        Exception ("Permission Denied.");
      }
    }
    @Submit [Student]{
      If( ! (team Contains e.Sender) )  {
        Exception ("Permission Denied.");
      }
      Lock(reportID);
      Email(supervisor, "Submitted");
      To(Pending);
    }
  }
  State Pending {
    @Accept[Professor] {
      If( e.Sender != supervisor ) {
        Exception ("Permission Denied.");
      }
      Publish(reportID);
      To(Published);
    }
    @Reject[Professor] {
      If( e.Sender != supervisor ) {
        Exception  ("Permission Denied.");
      }
      UnLock(reportID);
      Email(team, "Rejected");
      To(Draft);
    }
  }
  Final State Published;
}
```

Listing 3.2: The specification of the sample project report co-authoring collaboration

In this collaboration specification, we declare four fields for storing persistent data needed in the collaboration. The fields projectID and reportID keep the value of the parameters with the same names in Create event. The fields team and supervisor respectively declare the identifiers of the students who work in the project and the professor who acts as the supervisor. As the values for these fields are not initially provided and they are not also passed as the parameters of events directly, we need to identify these values. In the entry of the collaboration, we identify the team members using the Member relation, and identify the supervisor using the Find method and the Supervise relation defined in the configuration. Afterward, the field team contains a list of all students participating in that specific project, and the field supervisor contains the professor supervising that project.

Once instantiated, the collaboration goes to Draft state in which we defined two event handlers. The Edit event handler does not perform any action but checks whether the person who triggered the Edit event is a student and a member of the team; if not, it calls the Exception method to register an unauthorized event trigger. The Submit event-handler, in addition to performing the authorization check, changes the state into Pending, informs the supervisor about the state change, and locks the report.

In the Pending state, we also have two event handlers, both related to supervisor actions. If supervisor rejects the report by triggering the Reject event, the collaboration returns to the Draft state, the team is notified by email, and the report is unlocked for further editing. On the other hand, if the supervisor triggers the Accept event, the collaboration changes its state to Published and it calls the Publish method which finalizes the report.

Let us now consider a slightly more complex version of the project report co-authoring collaboration. In this version, after the submission of a project report, a set of checks should be performed to ensure that the report is ready for the final review by the supervisor. These checks include text check, figures check, and references check. There is no specific execution order among these checks and the order in which the checks are performed does not matter; it is only important that all these checks are performed before the report is sent to the supervisor. Since these checks are relevant to various types of documents and collaborations, we chose to implement them as a separate collaboration named DocumentCheck collaboration. Now, any collaboration can use this collaboration as a sub-collaboration by referencing to it. Since the order of checks to be performed in DocumentCheck collaboration does not matter, we implemented this collaboration using the rule-based style. The specification of DocumentCheck collaboration is presented in the following.

```
Collaboration RuleBased DocumentCheckCollaboration {
  // Field Declarations
  Boolean TextChecked;
  Boolean FigureChecked;
  Boolean ReferenceChecked;

  // Entry Specifications
  Entry Start {
    TextChecked = False;
    FigureChecked = False;
    ReferenceChecked = False;
  }

  // Event-handler Specifications
  @TextCheck {
    TextChecked = True;
    If (TextChecked And FigureChecked And ReferenceChecked ) {
      Trigger(Checked());
      TextChecked = False;
      FigureChecked = False;
      ReferenceChecked = False;
    }
  }
  @FigureCheck {
    FigureChecked = True;
    If (TextChecked And FigureChecked And ReferenceChecked ) {
      Trigger(Checked());
      TextChecked = False;
      FigureChecked = False;
      ReferenceChecked = False;
    }
  }
  @ReferenceCheck {
    ReferenceChecked = True;
    If (TextChecked And FigureChecked And ReferenceChecked ) {
      Trigger(Checked());
      TextChecked = False;
      FigureChecked = False;
      ReferenceChecked = False;
    }
  }
}
```

Listing 3.3: The specification of the sample document check collaboration

The DocumentCheck collaboration has three fields, TextChecked, FigureChecked and ReferenceChecked, each of them indicates whether a particular check has been performed on the document or not. The collaboration is initiated when it receives the Start event from the parent collaboration; then, the collaboration initializes all the fields to False. Upon completion of a check on the document, an event indicating the type of check is sent to the collaboration and the value of the corresponding field is set to True. When all fields are evaluated to True, signifying that all the checks have been done, the collaboration triggers Checked event to inform the parent about the completion of the collaboration.

In order to employ the DocumentCheck collaboration, in our example of the project report co-authoring collaboration, we first need to add the definition of the events used in the DocumentCheck collaboration in the configuration specification. There are five new events, namely Start, TextCheck, FigureCheck, ReferenceCheck, and Checked. Since the addition of these events to the configuration specification is straightforward and we described it before, we do not provide the new configuration specification here. Then, we should modify the project report co-authoring collaboration specification in such a way that it can interact with the DocumentCheck collaboration.

Figure 3.3 demonstrates the statechart of the new ReportingCollaboration. The most significant modification of the collaboration is the addition of a new state, named Checking, to the specification. The collaboration enters the Checking state upon the submission of the report, and stays in this state while waiting for all checks to be done, TextCheck, FigureCheck, and ReferenceCheck. Then it enters the Pending state and continues its routine as the previous version of the ReportingCollaboration.

The specification of the new ReportingCollaboration is provided in Figure 3.3. In order to em-

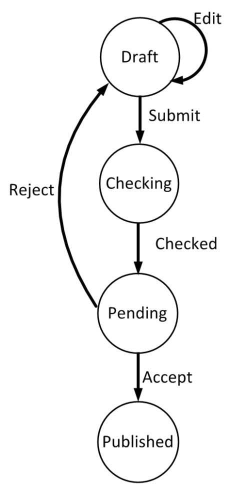

Figure 3.3: The statechart diagram of the updated sample project report co-authoring collaboration

phasize the modifications and also make the specification more clear, we did not include the lines which are not changed since the earlier version of the collaboration, but instead we placed "..." to indicate it. In the ReportingCollaboration, the fields were not modified; however, a sub-collaboration, named checkWf, is declared which refers to an instance of the DocumentCheckCollaboration specified before. Actually, the interaction of the ReportingCollaboration and DocumentCheckCollaboration is done through this declared sub-collaboration. In the Entry of the ReportingCollaboration, a Start event is triggered on checkWf which results in initiation of this subcollaboration. As it was explained before, a new state should be added to the specification of the ReportingCollaboration, named Checking. The collaboration enters Checking state upon its submission and stays in this state while waiting for completion of all checks in checkWf. In addition, the collaboration listens to TextCheck, FigureCheck, and ReferenceCheck events and directs them to checkWf as long as the Checking state is active.

```
Collaboration StateBased ReportringCollaboration  {
  // Fields
  ...

  // Sub-collaborations
  DocumentCheckCollaboration checkWf;

  // Entry Specifications
  Entry Create  {
    ...
    checkWf.Trigger(Start());
    To(Draft);
  }

  // State Specifications
  State Draft {
    @Edit [Student]{
      ...
    }
    @Submit [Student]{
      ...
      To(Checking);
    }
  }
  State Checking {
    @TextCheck{
      checkWf.Trigger(e);
    }
    @FigureCheck{
      checkWf.Trigger(e);
    }
    @ ReferenceCheck {
      checkWf.Trigger(e);
    }
    @ wf.Checked {
      To(Pending);
    }
  }
  State Pending {
    ...
  }
  Final State Published;
}
```

Listing 3.4: The specification of the updated sample project report co-authoring collaboration

Now that we have explained the two styles for writing collaboration specification, i.e., the state-based and the rule-based style, and have provided corresponding examples, let us explain why both of these styles are required, or rather, why each one of them is not appropriate for writing all the collaboration specifications. We used the state-based style for specifying the ReportingCollaboration and the rule-based style for specifying the DocumentCheckCollaboration; this choice argues for our belief that each of these styles is more suitable for specifying the corresponding collaboration. The state-based style is more suitable than the rule-based style for the ReportingCollaboration, because there is a logical dependency among the steps of the collaboration; this logical dependency is captured by the corresponding states and their sequencing. Had we chosen the rule-based style, the states would have to be "simulated" using variables, which would require a lot of variable evaluation and checking and would result in a much harder to write and understand specification. On the other hand, the rule-based style is more natural for specifying the DocumentCheckCollaboration, since the three checks can be performed in any arbitrary order. If we had chosen to write the source-checking collaboration in the state-based style, we would have needed to define nine different states to represent the possible combinations of the order in which the checks might be completed. Each of the states would have a couple of event-handlers but all of the event-handlers would almost do the same actions.

Clearly, these two styles for writing collaboration specifications have different usage scenarios and they actually complement each other. In our system, the collaboration editor supports both styles, so users can select which style they want to employ for writing each collaboration specification based on the properties of the collaboration.

## 3.2 Language Specification

In order to declare a collaboration in our proposed language, one needs to know how the structure of the collaboration model should be, and how these elements can communicate with each other to provide the intended behavior. Therefore, the collaboration language should specify what words and symbols (lexical elements) are allowed in the language, how these words can be arranged into statements (syntactic rules), and what rules and constraints should be considered regarding each of these statements while coding the collaboration model (semantic rules). In this section, we provide a complete specification of our language. For this purpose, we introduce the lexical elements of the language, provide the grammatical rules of the language as its syntax, and describe the semantics rules of the language which also includes its constraints in defining a collaboration.

### 3.2.1 Lexical Elements

Lexical elements are the smallest building blocks of a language, and are known as characters or groupings of characters that may appear in a source file. **Tokens** are the smallest meaningful elements of a program. Therefore, any program written in the target language, i.e. any collaboration specified in our collaboration language, is actually seen as a sequence of tokens. A token can be a keyword, identifier, literal, punctuator or an operator, and should be separated from other tokens with white spaces or comments.

**Keywords** are language specific words that have special meanings in the language. Keywords are words reserved by the language for special use, and no other elements in the program can have the same word as the keywords, for instance one cannot name a method as "Collaboration" since this is a keyword of the language and can only be used as the start of a collaboration declaration. Keywords are case-sensitive, as the language is case sensitive, and each indicates a specific meaning in the program. Table 3.1 shows the list of all keywords defined in our collaboration language.

**Identifiers** are the words selected as the names of elements that are defined in the program. In other words, whenever an element, such as a collaboration, sub-collaboration, state, event, role, relation, field or variable, is being defined, it should be named with an identifier. Syntactically, in our collaboration language, an identifier is a case-sensitive word that starts with an underscore or a letter and can be followed by any number of letters, underscores and digits. Note that the identifiers, i.e. the names selected for the program elements, should not be the same as any of the keywords.

**Literals** are constant values that occur in a program as values and cannot be changed. In our collaboration language the literals are Integer, String and Boolean values that can be used in expressions and statements, e.g., while assigning a value to a variable. Every literal has a data type, which is either Integer, String or Boolean in our language. An integer literal is either 0 or a sequence of digits 0 to 9 which cannot start by 0; a string literal is the character "(double quote), followed by any sequence of characters followed by the character " ; a Boolean literals can be true or false. There is no literal defined in the collaboration language for types Time, Strings, User, and Users.

| **Keyword** | **Meaning** |
|---|---|
| Collaboration | Indicates the start of a collaboration declaration. |
| Entry | Indicates the start of an entry point for a collaboration. |
| State | Indicates the start of a state declaration. |
| Event | Indicates the start of an event declaration in the configuration specification. |
| Role | Indicates the start of a roll declaration in the configuration specification. |
| Relation | Indicates the start of a relation declaration in the configuration specification. |
| Integer | Indicates the start of an integer variable declaration. |
| Boolean | Indicates the start of a Boolean variable declaration. |
| String | Indicates the start of a String variable declaration. |
| Strings | Indicates the start of a Strings variable declaration. |
| Time | Indicates the start of a Time variable declaration. |
| User | Indicates the start of a User variable declaration. |
| Users | Indicates the start of a Users variable declaration. |
| If | Indicates the start of a conditional control statement. |
| Else | Resumes a conditional statement with an opposite logical value. |
| While | Indicates the start of a While loop statement. |
| Foreach | Indicates the start of a Foreach statement that iterates over a list. |
| in | Separates the iterator and the list in a Foreach statement. |
| On | Indicates a time event handler activating on a specific time. |
| All | A predefined method to return all the users of a specific role. |
| Is | A predefined method to check whether a user has a role or not. |
| Find | A predefined method to find the entities which matches a query. |
| e | Indicates the start of an event parameter access. |
| Sender | A predefined parameter for events related to the sender of the event. |
| null | Indicates a null value. |
| True | Indicates the true value as a Boolean literal. |
| False | Indicates the true value as a Boolean literal. |
| Trigger | A predefined method used for triggering an event. |
| To | A predefined method used for indicating a state change. |
| POST | A predefined method used for calling a service. |
| GET | A predefined method used for calling a service. |
| Exception | A predefined method used for indicating the occurrence of an exception. |
| And | Binary logical operators &&, a.k.a. AND. |
| Or | Binary logical operators $\|\|$, a.k.a. OR. |
| Final | Indicates that the following State is a final state. |
| Terminate | A predefined method used for deactivating a rule-based collaboration. |
| WfCreator | Refers to the user who created the collaboration instance. |
| WfId | Refers to the identifier of the collaboration instance. |
| Contains | A predefined method to check whether a value exists in a collection or not. |
| StateBased | Indicates that the collaboration is programmed in the state-based style. |
| RuleBased | Indicates that the collaboration is programmed in the rule-based style. |

Table 3.1: The keywords of our collaboration language

| Symbol | Meaning |
|---|---|
| @ | Indicates an event handler activating at occurrence of a specific event. |
| { } | Groups a set of lines of code. |
| ( ) | Groups a set of different syntactical elements. |
| [ ] | Indicates a role in an entry declaration. |
| /∗ ∗/ | Indicates a comment. |
| // | Indicate a single-line comment. |
| “ ” | Indicates a string literal. |
| ; | Indicates the end of a line. |
| : | Indicates the required URL(s) in role, relation and service declarations presented in the following. |
| , | Separates method parameters in service declarations, separates URLs in role and relation declaration, and separates expressions in an expression list. |
| . | Accesses the parameters of a construct. |
| ? | Indicates the target side of a relation in the query of a find Role&Relation expression. |
| = | Assigns the right-hand side value to the left-hand side variable. |
| + − ∗ / | Binary Arithmetic operators for numerical expressions. |
| ∗ | Identifies that the parameter of an event is mandatory. |
| + | An operator for concatenating two String variables, adding a String (or User) to a collection of Strings (or Users) and appending two collections of Strings (or Users). |
| > < | Binary comparators that mean larger than and less than respectively. |
| == | Binary comparator that means Is equal to. |
| ! = | Binary comparator that means Is not equal to. |
| ! | Unary logical operator NOT. |

Table 3.2: The operators and punctuators of our collaboration language

**Punctuators** are characters or symbols that form the statements and expressions in the program, and therefore have specific syntactic and semantic meanings; For example, a semicolon “;” is a punctuator that shows the end of a line in our collaboration language. Table 3.2 shows the list of punctuators that may occur in a program written in our language.

**Operators** are tokens employed to perform a manipulation on different data types in the program. Operators can be either symbols (e.g. ==, !=, !) that are in fact punctuators, or reserved words (e.g. Contains, Is, And, Or) that are keywords. However, since operators need operands to manipulate and calculate a result, we put them in a separate group of tokens to emphasize on their syntactic and semantic meaning. In our collaboration language operators can be one of the followings:

- Logical operators, including And, Or, !
- Arithmetic operators, including +, -, *, /
- String operators, including +
- Collection operators, including +, -, Contains
- Relational operators, including >, <, ==, !=
- Other operators, including Is

Other than meaningful tokens, the language supports white spaces and comments that have no meaning except for separating the tokens. Compiler ignores these meaningless units during the preprocessing since they add no logic to the program. **White spaces**, e.g., space, new line, and tabulator, do not have any meaning but are used to separate the tokens from each other, and possibly to make the code more readable. **Comments** can occur in source file in order to add explanations about the program; however, during preprocessing, the compiler replaces comments by a single space character. Syntactically, comment is the string /* (slash, asterisk), followed by any sequence of characters (including new lines), followed by the string */ and can occur anywhere the language allows white spaces. Hence we cannot have nested comments; therefore, further */ within a comment are ignored and the comment ends at the first occurrence of */. In addition, for single-line comments string // (slash, slash) can be placed in front of a line, so the line is treated as a piece of comments.

### 3.2.2 Syntax

The syntax of a programming language includes the set of rules indicating how the lexical elements should be combined together and make declarations and statements, so that the resulted program be correctly structured. Accordingly, the syntactic specification of a language is actually the grammatical rules designed for the domain specific language.

Since we have different structures for programming collaborations and their configuration rules, the grammar of our collaboration language consists of two separate grammars: (1) a grammar for writing Collaboration Specifications, (2) a grammar for writing the Configuration Specification. However, these two grammars are closely connected to each other as elements defined in a configuration specification are used in related collaboration specifications.

In this section the grammar of our collaboration language has been presented in BNF (Backus-Naur Form). In presenting this grammar we used the notations which are explained as follows:

- $\rightarrow$ means the left-hand side can be replaced by the right hand side set.
- Non-terminals are printed in bold typewriter font. e.g. **Expression**.
- Terminals are shown inside “”, e.g. “Collaboration”.
- The identifiers are written in bold and Italic typewriter font, e.g. ***CollaborationName***.
- X ? means zero or one occurrence of X.
- X * means zero, one or more occurrences of X.
- | indicates alternatives.
- ( ) groups multiple syntactical elements.

The grammar for writing collaboration specifications is provided in Table 3.3.

| | | |
|---|---|---|
| **CollaborationSpecification** | ⟶ | “Collaboration” **CollaborationStyle** ***CollaborationName*** “{” |
| | | **FieldDeclaration*** |
| | | **SubCollaborationDeclaration*** |
| | | **EntryDeclaration*** |
| | | ( **State-basedLogic** \| **Rule-basedLogic** ) “}” |
| **CollaborationStyle** | ⟶ | “StateBased” \| “RuleBased” |
| **FieldDeclaration** | ⟶ | **Type** ***FieldName*** “;” |
| **Type** | ⟶ | “Boolean” |
| | | \| “Integer” |
| | | \| “String” |
| | | \| “Strings” |
| | | \| “Time” |
| | | \| “User” |
| | | \| “Users” |
| **Sub-CollaborationDeclaration** | ⟶ | ***CollaborationName InstanceName***“;” |
| **EntryDeclaration** | ⟶ | “Entry” ***EventName*** (“[” **RoleList** “]”)? Block |
| **State-basedLogic** | ⟶ | **StateDeclaration*** |
| **Rule-basedLogic** | ⟶ | (**EventHandler** \| **TimeHandler**)* |
| **StateDeclaration** | ⟶ | “Final” “State” **StateName** “;” |
| | | “State” **StateName** “{” (**EventHandler** \| **TimeHandler**)*“}” |
| **EventHandler** | ⟶ | “@” ***EventName*** (“[” **RoleList** “]”)? **Block** |
| **TimeHandler** | ⟶ | “On” **FieldName Block** |
| **Block** | ⟶ | “{” **Statement*** “}” |
| **Statement** | ⟶ | **EmptyStatement** |
| | | \| **ExpressionStatement** |
| | | \| **IfStatement** |
| | | \| **WhileStatement** |
| | | \| **ForeachStatement** |
| | | \| **VariableDeclaration** |
| | | \| **Assignment** |
| | | \| **EventTrigger** |
| | | \| **StateChange** |
| | | \| **TerminateCollaboration** |
| | | \| **Exception** |
| **EmptyStatement** | ⟶ | “;” |
| **ExpressionStatement** | ⟶ | **Expression**“;” |
| **VariableDeclaration** | ⟶ | **Type** ***VariableName***“;” |
| **IfStatement** | ⟶ | “If” “(” **Expression**“)” **Block** (“Else” **Block**)? |
| **WhileStatement** | ⟶ | “While” “(” **Expression**“)” **Block** |
| **ForeachStatement** | ⟶ | “Foreach” “(” ***VariableName*** “in” **Expression**“)” **Block** |
| **AssignmentStatement** | ⟶ | ( **FieldName** \| **VariableName**) “=” **Expression** “;” |
| **EventTrigger** | ⟶ | “Trigger” “(” (***InstanceName*** “.”)? ***EventName*** “(” **ExpressionList**? “)” “)” “;” |
| **StateChange** | ⟶ | “To” “(” ***StateName*** “)” “;” |
| **TerminateCollaboration** | ⟶ | “Terminate” “;” |
| **Exception** | ⟶ | “Exception” “(” **Expression** “)” “;” |
| **RoleList** | ⟶ | ***RoleName*** (“,” ***RoleName***)* |
| **ExpressionList** | ⟶ | **Expression** (“,” **Expression**)* |
| **Expression** | ⟶ | Literal |
| | | \| **ReferenceExpression** |
| | | \| **LogicalExpression** |
| | | \| **RelationalExpression** |

| | | |
|---|---|---|
| | | \| **CollectionExpression** |
| | | \| **ArithmaticExpression** |
| | | \| **StringExpression** |
| | | \| **Role&RelationExpression** |
| | | \| **ServiceInvokation** |
| | | \| “(” **Expression** “)” |
| **Literal** | ⟶ | <INTEGER_LITERAL> |
| | | \| <STRING_LITERAL> |
| | | \| <BOOLEAN_LITERAL> |
| | | \| “null” |
| **ReferenceExpression** | ⟶ | ***FieldName*** |
| | | \| ***VariableName*** |
| | | \| “e.” ***ParameterName*** |
| | | \| “e.Sender” |
| | | \| “WfCreator” |
| | | \| “WfId” |
| **LogicalExpression** | ⟶ | **Expression** (“And” \| “Or”) **Expression** |
| | | \| “!” **Expression** |
| **RelationalExpression** | ⟶ | **Expression** ( “<” \| “>” \| “==” \| “!=”) **Expression** |
| | | \| **ReferenceExpression** “Contains” **Expression** |
| **CollectionExpression** | ⟶ | **ReferenceExpression** (“+” \| “-”) **Expression** |
| **ArithmeticExpression** | ⟶ | **Expression** (“+” \| “-” \| “*” \| “/”) **Expression** |
| **StringExpression** | ⟶ | **Expresion** “+” **Expression** |
| **Role&RelationExpression** | ⟶ | **ReferenceExpression** “Is” ***RoleName*** |
| | | \| “All” ***RoleName*** |
| | | \| **ReferenceExpression** ***RelationName*** **ReferenceExpression** |
| | | \| “Find” “(” “?” ***RelationName*** **ReferenceExpression** “)” |
| | | \| “Find” “(”**ReferenceExpression** ***RelationName*** “?” “)” |
| **ServiceInvocation** | ⟶ | ***ServiceName*** “(” **ExpressionList**? “)” |
| ***CollaborationName*** | ⟶ | <IDENTIFIER> |
| ***FieldName*** | ⟶ | <IDENTIFIER> |
| ***InstanceName*** | ⟶ | <IDENTIFIER> |
| ***EventName*** | ⟶ | <IDENTIFIER> |
| ***RoleName*** | ⟶ | <IDENTIFIER> |
| ***StateName*** | ⟶ | <IDENTIFIER> |
| ***VariableName*** | ⟶ | <IDENTIFIER> |
| ***ServiceName*** | ⟶ | <IDENTIFIER> |
| ***ParameterName*** | ⟶ | <IDENTIFIER> |

Table 3.3: The grammar of collaboration specifications

The grammar for writing configuration specification is presented in Table 3.4.

| | | |
|---|---|---|
| **ConfigurationSpecification** | ⟶ | (**ConfigurationMember**)* |
| **ConfigurationMember** | ⟶ | **EventDeclaration** |
| | | \| **RoleDeclaration** |
| | | \| **RelationDeclaration** |
| | | \| **ServiceDeclaration** |
| **EventDeclaration** | ⟶ | “Event” ***EventName*** “(” **EventParameterList**? “)” “;” |
| **RoleDeclaration** | ⟶ | “Role” ***RoleName*** “(” ***ParameterName*** “)” “:” **URL** “,” **URL** “;” |
| **RelationDeclaration** | ⟶ | “Relation” ***RelationName*** “(” **Parameter** “,” **Parameter** “)” “:” **URL** “,” **URL** “;” |

| | | |
|---|---|---|
| **ServiceDeclaration** | $\longrightarrow$ | **Type** (“POST” \| “GET”) ***ServiceName*** “(” **ParameterList**? |
| | | “)” “:” **URL** “;” |
| **ParameterList** | $\longrightarrow$ | **Parameter** (“,” **Parameter** )* |
| **EventParameterList** | $\longrightarrow$ | **EventParameter** ( “,” **EventParameter**)* |
| **EventParameter** | $\longrightarrow$ | **Parameter** (“*”)? |
| **Parameter** | $\longrightarrow$ | **Type** ***ParameterName*** |
| **Type** | $\longrightarrow$ | “Boolean” |
| | | \| “Integer” |
| | | \| “String” |
| | | \| “Strings” |
| | | \| “Time” |
| | | \| “User” |
| | | \| “Users” |
| **URL** | $\longrightarrow$ | $<$STRING_LITERAL$>$ |
| ***EventName*** | $\longrightarrow$ | $<$IDENTIFIER$>$ |
| ***RoleName*** | $\longrightarrow$ | $<$IDENTIFIER$>$ |
| ***ParameterName*** | $\longrightarrow$ | $<$IDENTIFIER$>$ |
| ***ServiceName*** | $\longrightarrow$ | $<$IDENTIFIER$>$ |

Table 3.4: The grammar of configuration specifications

### 3.2.3 Semantics

Semantics of a language include all the rules and constraints any program written in the target language should follow, unless the program is not implemented correctly. For instance, while assigning a value to a variable in an Assignment Statement the type of the left-hand side and the right-hand side of the assignment should be the same; therefore you cannot assign the value True to a variable of type Integer.

Most of the semantic rules applied in our collaboration language, specifically the ones about type consistencies in statements and expressions, are the same as the ones used in Java programming language. However, there are some domain specific rules regarding the declaration of collaborations and the configuration specification and their inner elements (e.g. roles, users, fields and variables). In this section we describe each of these domain specific elements and provide the language semantic rules for using each of them. Violating any of these rules in a program written in our collaboration language will result to a compilation error indicating that the program (either the collaboration or the configuration specification) is not modeled correctly.

#### Collaboration Specification

*Description:*

Collaboration Specification defines a collaboration as a complete unit. It is the highest level element of the collaboration language and encompasses the data and logic pieces of a collaboration. A Collaboration Specification is composed of a logic section in addition to some fields, sub-collaborations, and entries. The logic section can be written in either rule-based style or state-based style.

*Rules:*

- Collaborations must have distinct names in a system.
- The members of a collaboration specification should be placed in a pre-defined order: field declaration, sub-collaboration declaration, entry declaration, and logic.
- The Collaboration Style must match the style in which the logic section is written. In other words, if a collaboration specification is labeled as "StateBased", the logic section should be written in state-based style; and if the collaboration specification is labeled as "RuleBased", the logic section should be written in rule-based style.

**Field Declaration**

*Description:*

Fields act as the data elements of a collaboration. Every field is a data placeholder to which you can assign value; then, you can access the value whenever it is needed. The values of the fields are kept as persistent data. A field declaration is composed of a type and a name.

*Rules:*

- The fields of a collaboration must have distinct names.

**Sub-Collaboration Declaration**

*Description:*

Sub-Collaboration Declaration enables creation of instances of other collaborations. A Sub-Collaboration Declaration is composed of the name of a collaboration and a name for the instance.

*Rules:*

- The sub-collaborations of a collaboration must have distinct names.
- The Collaboration Name must point to a specified collaboration in the system.
- A collaboration cannot declare a sub-collaboration of its type. In other words, recursive initiation of sub-collaborations is not permitted.

**Entry Declaration**

*Description:*

Entry is a special type of event-handlers used for instantiating the enclosing collaboration. It is also responsible for initializing the fields and sub-collaborations of the corresponding collaboration instance. An Entry Declaration is mainly composed of the identifier of an event type, some roles, and a block of code. Receiving an event of the type for which the entry is waiting, the entry first checks whether the sender holds any of the roles indicated for the entry; if yes, the entry executes the following block of code; otherwise, the entry throws an exception. The execution of every collaboration instance always starts by execution of one of its entries; and when the instance is created, no entry can be executed on that instance afterward.

*Rules:*

- The Event Name must point to a defined event in the configuration.
- The Role List must point to some defined roles in the configuration.
- There cannot be two entries in a collaboration with same Event Names. In other words, no event should result in execution of more than one entry of a collaboration.
- There cannot be any pair of entry and event-handler in a collaboration with same Event Names. In other words, no event should result in execution of more than one entry or event-handler of a collaboration.
- There must exactly one State Change in the Block of an Entry Declaration.

**State Declaration**

*Description:*

In state-based collaboration specifications, which is labeled as "StateBased", States denote the significant phases through which collaborations passed in their life cycles. At any given point, a state-based collaboration instance is in one of its states. A state specifies the behavior of the collaboration when the collaboration is in that particular state. States are also responsible for indicating their next states, unless they are labeled as "Final". When a collaboration reaches a Final State, it gets deactivated and no longer listens to events. Every State Declaration is composed of a State Name and a logic section that is defined as a collection of Event-Handlers and Time-Handlers.

*Rules:*

- The States of a collaboration must have distinct names.
- State declaration can only be used in state-based collaborations.
- If the State is labeled as "Final", it must have no logic section.

**Event-Handler**

*Description:*

An Event-Handler involves some activates which are performed when a specific event is sent to the corresponding collaboration instance. An Event-Handler is mainly composed of the name of an event defined in the Configuration Specification, a list of roles as the only roles allowed to trigger the event, and a block of code. When the scope to which the Event-Handler belongs gets activated, the Event-Handler starts waiting for the specified event type. The targeted event may be received from the parent collaboration or sub-collaborations. Receiving an event of the type for which the entry is waiting, the Event-Handler first checks whether the sender holds any of the roles indicated for the Event-Handler; if yes, the Event-Handler executes the following block of code; otherwise, the entry throws an exception. When the scope to which the Event-Handler belongs gets deactivated, the event-handler also gets deactivated.

*Rules:*

- Before an Event-Handler can get activated, the collaboration instance to which the Instance Name refers should be initiated, i.e. one of its entries should be invoked.
- The Event Name must point to a defined event in the configuration.
- The Role List must point to some defined roles in the configuration.
- There cannot be two Event-Handlers in the same scope with same Event Names. In other words, no event should result in execution of more than one Event-Handler of the same scope.
- There cannot be any pair of entry and event-handler in a collaboration with same Event Names. In other words, no event should result in execution of more than one entry or event-handler of a collaboration.

**Time-Handler**

*Description:*

A Time-Handler involves some activates which are performed when a specific point of time is reached. Time-handlers are similar to Event-Handlers; but unlike Event-Handlers that listen for events, they wait for a specific amount of time. A Time-Handler is mainly composed of the name of a Time field and a block of code. When the scope to which the time-handler belongs gets activated, the time-handler reads the value of the indicated time field; when the time is reached, the time-handler executes the following block of code. When the scope to which the time-handler belongs gets deactivated, the time-handler also gets deactivated.

*Rules:*

- The Field Name must point to a field of type Time declared in the Field Declaration section.
- At any time, there should not be multiple active time-handlers waiting for the same point of time.

**Variable Declaration**

*Description:*

Variables are used for keeping temporary data in a scope. Every variable is a data placeholder to which you can assign value; then, you can access the value whenever it is needed. A variable is accessible form its scope and all the descendant scopes. As soon as a scope gets deactivated, all of the variables defined in the scope lose their values. A variable declaration is composed of a type and a name.

*Rules:*

- The name of a variable should be distinct among all the variable names declared in or accessible from the scope in which the variable is defined.

**If Statement**

*Description:*

If Statement is the main conditional structure of the collaboration language. An If Statement is mainly composed of a condition and a block of code. When the If Statement is executed, it evaluates the specified condition; if the condition is satisfied, the If Statement executes its following block of code. In addition to the main block of code, every If Statement can have another block of code which is executed when the condition is evaluated to false, i.e. the block of code programmed in the block following "Else" keyword.

*Rules:*

- The expression used as the condition in an If Statement must return a value of Boolean type.
- The expression used as the condition in an If Statement could be either a reference to a Boolean field or a logical expression of some fields.

**While Statement**

*Description:*

While Statement is one of the loop structures of the collaboration language. It can be seen as a repeating If Statement. A While Statement is mainly composed of a condition and a block of code. When the While Statement is executed, it evaluates the specified condition; if the condition is satisfied, the While statement executed its following block of code; this process repeats and the block of code gets executed as long as the condition is satisfied.

*Rules:*

- The expression used as the condition in a While Statement must return a value of Boolean type.
- The expression used as the condition in a While Statement could be either a reference to a Boolean field or a logical expression of some fields.

**Foreach Statement**

*Description:*

Foreach Statement is one of the loop structures of the collaboration language. It can be used for traversing items in a collection. As Users and Strings are the only collection types in the collaboration language, the Foreach Statement can only be used on instances of these types. A Foreach Statement is mainly composed of a variable name, a collection, and a block of code. When the Foreach Statement is executed, it iterates over the items of the specified collection. In each iteration, the Foreach Statement declares a variable using the provided name, assigns the current item of the collection to the variable, and executes its following block of code accordingly. If the type of the collection is Users, the type of the variable will be User; and if the type of the collection is Strings, the type of the variable will be String.

*Rules:*

- The Variable Name used in the Foreach Statement should be distinct among all the Variable Names declared in or accessible within the Foreach Statement.
- The expression used after "in" keyword in the Foreach Statement must return a value of Users type or Strings type.

**Assignment Statement**

*Description:*

Assignment Statements are used for assigning values to variables and fields. An Assignment Statement is mainly composed of a variable or a field followed by the = punctuator and an expression with the same type as left-hand side variable or field. When the Assignment Statement is executed, the expression is evaluated and its value is assigned to the variable or field.

*Rules:*

- The Field Name must point to a field declared in the Field Declaration.
- The Variable Name must point to a variable accessible from the scope to which the Assignment Statement belongs.

- Any assignment to a variable cannot be performed unless its Variable Declaration has already been executed.
- The expression used in the Assignment Statement must return a value of the same type as the field (or variable) to which the Field Name refers.

**Event Trigger**

*Description:*

Event Trigger enables sending events to other collaborations. It is responsible for creating an instance of the specified event type and triggering it on the parent collaboration or one of the sub-collaborations. An Event Trigger statement is mainly composed of the name of an event and a list of expressions. It also may include the name of a declared sub-collaboration. When the Event Trigger statement is executed, the Event Trigger creates an instance of the specified event and sets the event's parameters according to the Expression List. If the name of a sub-collaboration is indicated, the event is sent to that sub-collaboration; otherwise, the event is sent to the parent collaboration according to the hierarchy of collaboration instances.

*Rules:*

- The Instance Name must point to a collaboration instance declared in the Sub-Collaboration Declaration section.
- The Event Name must point to a defined event in the configuration.
- The number of expressions in the Expression List must be the same as the number of parameters of the specified event.
- Each expression in the Expression List must have the same return type as its corresponding parameter of the event.

**State Change**

*Description:*

State Change enables state-transition in state-based collaborations. It is responsible for changing the state of the collaboration to the specified state. In addition to state-transition, execution of a State Change statement will result in deactivation of all the handlers (i.e. Event-Handler, Time-Handler) defined in the old state and activation of all the handlers defined in the new state.

*Rules:*

- The State Name must point to a defined state in the collaboration.
- State Change can only be used in state-based collaborations.

- State Change can only be used as the last statement of any Event-Handler or Time-Handlers.
- State Change cannot be used in Final States.

**Terminate Collaboration**

*Description:*

Terminate Collaboration is used for indicating the completion of a rule-based collaboration. In rule-based collaborations, executing a Terminate Collaboration statement has the same impact as reaching a Final state in state-based collaborations. When a Terminate Collaboration statement is executes, the collaboration gets deactivated and no longer reacts to events.

*Rules:*

- Terminate Collaboration can only be used in rule-based collaborations.
- Terminate Collaboration can only be used as the last statement of any Event-Handler, or Time-Handlers.

**Exception**

*Description:*

Exception is a type of statement used for indicating that an exception has happened in a collaboration. It is mainly composed of only an expression which provides a description about the exception. When an Exception statement is executed, the Exception evaluates the expression and publishes the value. The published values by exceptions are logged and can be used for analyzing collaborations and debugging the system. After executing an Exception statement, the collaboration does not keep on executing the next statements, but terminates the execution and stops waiting for events.

*Rules:*

- The expression provided for any Exception must return a value of String type.

**Reference Expression**

*Description:*

Reference Expression is used for accessing the value of a field, variable, or event parameter. Fields and variables can be referred by their names; however, in order to refer event parameters, the prefix “e.” should be added before their names; for example e.projectID for the event Create defined in the example presented in Section 3.1 refers to the ProjectID parameter defined for the Create event. Furthermore, every collaboration instance has two generic read-only fields: WfId and WfCrator. WfId is a field of type String which contains the identifier of the collaboration instance, and WfCreator is a field of type User which refers to the user who created the collaboration instance. These fields are initiated when the collaboration instance is created. Similarly, every event has a generic parameter named Sender. The Sender is a parameter of type User which refers to the user who created the event.

*Rules:*

- Reference Expressions to parameters of events can be only used in Event-Handlers and Entries.
- The Parameter Name must point to a parameter of the corresponding event.
- The Field Name must point to a field declared in the collaboration.
- The Variable Name must point to a variable declared in or accessible from the place where the Reference Expression is used.

**Logical Expression**

*Description:*

Logical Expression is used for working with Boolean values. It applies a logical operator on one or two Boolean operands. The return type of Logical Expressions is also Boolean. The logical operators allowed in our language are And, Or, and !.

*Rules:*

- For And and Or operators, the return types of both operands must be Boolean.
- For ! operator, the return type of the operand must be Boolean.

**Relational Expression**

*Description:*

Relational Expression is used for checking a relation between two values by using relational operators. The return type of Relational Expressions is Boolean. The relational operators are $>$, $<$, ==, !=, and Contains. Contains operator checks whether a collection contains an item or not. In addition, it should be noted that == and != operators can also be used on collections; in such case, they check whether two collections contain the same set of items or not.

*Rules:*

- For $>$ and $<$ operators, the type of both the left-hand side and the right-hand side operands must be Integer.

- For == and != operators, the type of both the left-hand side and the right-hand side operands must be the same.
- For Contains operator, the type of the left-hand side operand must be either Users or Strings. If it is of type Users, the type of the right-hand side operand must be User; otherwise, the type of the right-hand side operand must be String.

**Arithmetic Expression**

*Description:*

Arithmetic Expression is used for working with Integer values. It applies an arithmetic operator on two Integer operands. The return type of Arithmetic Expressions is also Integer. The arithmetic operators are +, -, *, /.

*Rules:*

- For all the operators, the types of both the left-hand side and the right-hand side operands must be Integer.

**String Expression**

*Description:*

String Expression is used for working with String values. It applies a String operator on two String operands. The return type of String Expressions is also String. The only String operator is + which concatenates two Strings and builds a new one.

*Rules:*

- For the + operator, the types of both the left-hand side and the right-hand side operands must be String.

**Collection Expression**

*Description:*

Collection Expression is used for working with collections, i.e. Strings and Users. The return type of Collection Expression is either Strings or Users depending on the types of the operands. The collection operators are + and -. The + operator is used to add some values to a collection. The - operator is used for remove some values from a collection. It should be mentioned that adding to or removing form a collection will result in creation of a new collection and the original collection will be unchanged.

*Rules:*

- For both operators, the type of the left-hand side operand must be Strings or Users. If the type of the left-hand is Strings, the type of the right-hand must also be String or Strings; otherwise, the return type of the right-hand side must be User or Users.

**Role&Relation Expression**

*Description:*

Role&Relation Expression enables using the roles and relations defined in the provided configuration specification. There are five types of Role&Relation Expressions. The first type is composed of a value of User type and the name of a role connected with “Is” operator; it checks whether the provided user has the specified role or not using the first URL specified in the Role Declaration; for example the expression e.Sender Is Student Returns true if the role of the Sender of the triggered event is Student and return false otherwise. The second type is composed of “All” operator in addition to the name of a role; it returns a Users value containing all the users who have the specified role not using the second URL specified in the Role Declaration; for example the expression All Students return a Users collection of all people having the role Student. The third type is composed of two Reference Expressions connected with the name of a relation, and it checks whether the specified relation is held between the provided Reference Expressions or not; for example the expression userX Supervise projectX returns true if userX is the supervisor of projectX and return false otherwise. The fourth and fifth types are Find Expressions composed of “Find” operator, the ? punctuator, the name of a relation, and a Reference Expression. Both of these expressions return all the values which hold the specified relation with the provided Reference Expression. The fourth type, in which ? punctuator is placed before the name of the relation, returns the values which satisfy the relation as the left-hand side value; for example the expression Find( ? supervise projecID) returns a Users collection of all users that supervise the project with the identifier of projecID. The fifth type, in which the ? punctuator is placed after the name of the relation, return the value which satisfy the relation as the right-hand side value; for example the expression Find( userX supervise ?) returns a Strings collection of all projects that are being supervised by userX. In order to output the return values, Role&Relation Expressions employ Role Declarations and Relation Declarations. For this purpose, every Role&Relation Expression makes a service call, i.e. sends a GET/HTTP request, using the relevant URL provided by the corresponding Role Declaration or Relation Declaration; then, it outputs the response as the return value.

*Rules:*

- The return type of the Reference Expressions used in Is operator must be User.
- The Role Name must point to a defined role in the Configuration Specification.
- The Relation Name must point to a defined relation in the Configuration Specification.

- The type of the first parameter of the Reference Expression used before a Relation Name must be the same as the type of the first parameter of the corresponding Relation Declaration.
- The return type of the Reference Expression used after a Relation Name must be the same as the type of the second parameter of the corresponding Relation Declaration.

**Service Invocation**

*Description:*

Service Invocation enables using the services declared in the Configuration Specification. A Service Invocation expression is mainly composed of the name of a declared service and a list of parameters. When a Service Invocation expression is executed, it invokes the corresponding service with the provided parameters; then, it returns a value of the type indicated in the Service Declaration as the return type.

*Rules:*

- The Service Name must point to a defined service in the configuration specification.
- The number of expressions in the Expression List must be the same as the number of the parameters of the service to which the Service Name points.
- Each expression in the Expression List must have the same return type as its corresponding parameter of the service to which the Service Name points.

**Configuration Specification**

*Description:*

Configuration Specification defines all elements that are needed to interact with collaborations. In particular, events are incoming channels for the collaboration by which external components send messages to collaborations; while roles, relations, and services are the means by which the collaboration responds to events or receives additional information. Accordingly, a Configuration Specification consists of Event Declarations, Role Declarations, Relation Declarations, and Service Declarations.

*Rules:*

- In the system, exactly one Configuration Specification should be provided.
- At least, one Event Declaration must be defined in Configuration Specification; otherwise, there is no way to interact with collaborations.

**Event Declaration**

*Description:*

Event Declaration is used for defining the events of the collaboration system. In fact, events should be declared in the configuration specification in order to be used in Event-Handlers and Entry Declarations of the Collaboration Specifications. An Event Declaration is mainly composed of a name for the event and a list of parameters. The list of parameters indicates the data elements of the event. Each parameter in the list of parameters may have a * punctuator which identifies that the valuation of this parameter is mandatory at event creation time. Therefore, if any of the mandatory parameters is "null" when the event is triggered, the system throws an exception indicating that.

*Rules:*

- Events must have distinct Event Names.

**Role Declaration**

*Description:*

Role Declaration is used for defining the roles of users in the system. In fact, every role used in the collaboration specification should be first declared in the configuration specification. A Role Declaration is mainly composed of a name for a role, a list of Parameter Declarations, and two URLs. The first URL points to the REST service which checks whether a user has the role or not. To achieve that, a Parameter Name is also needed in the service call and checking whether a user has a role, this name will refer to a specific user while calling the service within a collaboration. The second URL points to the REST service which returns all the users who have that role. The URLs and parameter are used by Role&Relation Expressions to output the intended results, i.e. a Boolean value indicating a user has the role, or a list of users with that specific role.

*Rules:*

- Roles must have distinct Role Names.
- The Parameter of a Role Declaration must be of type User.

**Relation Declaration**

*Description:*

Relation Declaration is used for defining the relations between entities in the system. In fact, every relation used in the collaboration specification should be first declared in the configuration specification. A Relation declaration is mainly composed of the name for a relation, two parameters, and two URLS. The first URL points to the REST service which checks whether the relation is held between two entities, and the second URL points to the REST service which returns all the entities which satisfy the relation when the other entity is provided. The first parameter identifies the type and name of the left-hand side entity of the relation; similarly, the second parameter identifies the type and the name of the right-hand side entity of the relation. The URLs and parameters are used by Role&Relation Expressions to provide the intended return value. In fact, the two URLs determine the address of the REST services, and the parameters define the name of the parameters of service calls and also return types of Role&Relation Expressions.

*Rules:*

- Relations must have distinct Relation Names.
- The two parameters of each Relation Declaration must have different Parameter Names.
- The type of each parameter of a Relation Declaration can be either String or User.

**Service Declaration**

*Description:*

Service Declarations refer to REST services on the network. These services are required by the collaborations in order to complete their tasks. In fact, every service used by Service Invocations in the Collaboration Specifications should be first declared in the configuration specification. A Service Declaration is mainly composed of a return type, a HTTP method type, a name for the service, a list of parameters, and a URL. The URL points to the REST service we want to use; the list of parameters defines the parameters of the REST service calls; the HTTP method type indicates that the service call should use either GET/HTTP or POST/HTTP method; and the return types specifies the return type of the Service Invocations which use this service.

*Rules:*

- Services must have distinct Service Names.
- The Parameters of a Service Declaration must be of type User.
- The two parameters of Service Declaration must have different Parameter Names.

**URL**

*Description:*

URL is a value of type String which should refer to a service on the network.

*Rules:*

- URLs should be valid according to the standard URL format.

# Chapter 4

# The Software Framework

Having developed a language for specifying collaborations, the task becomes to develop a corresponding software system to support the specification and enactment of resource-centric collaborative activities. In fact, we are aiming to build a comprehensive toolset to be integrated with multi-user, multi-tool resource-processing environments in order to enable them to support resource-centric collaborations. In other words, we want to build neither a resource-processing environment nor any tools to be used directly by end-users, but a behind-the-scenes supporting system which leverages the capacities of any resource-processing environment to be more than just an editing tool but a process-aware environment which coordinates user collaborations.

The system consists of a set of software tools to specify collaborations and to manage their instances at run time, including the interaction of these collaboration instances with the users' activities in the context of other external systems. The main components of our collaboration-management system and their interactions are shown in Figure 4.1 and are described in detail below.

## 4.1 The Collaboration Engine

The collaboration engine is the fundamental component of the collaboration system, which enacts the collaborations at run-time. The collaboration engine is responsible for:

- Instantiating collaboration instances, according to provided specifications;
- Delivering events to the relevant collaboration instances;
- Triggering events on collaboration instances;
- Facilitating the execution of collaboration instances; and
- Managing collaboration instances through their life-cycle.

The collaboration engine must support all the constructs defined in the collaboration language, i.e., it should be able to check logical conditions, validate user access, invoke web-services, receive events and forward them to the relevant instances and manage the values of the various instances.

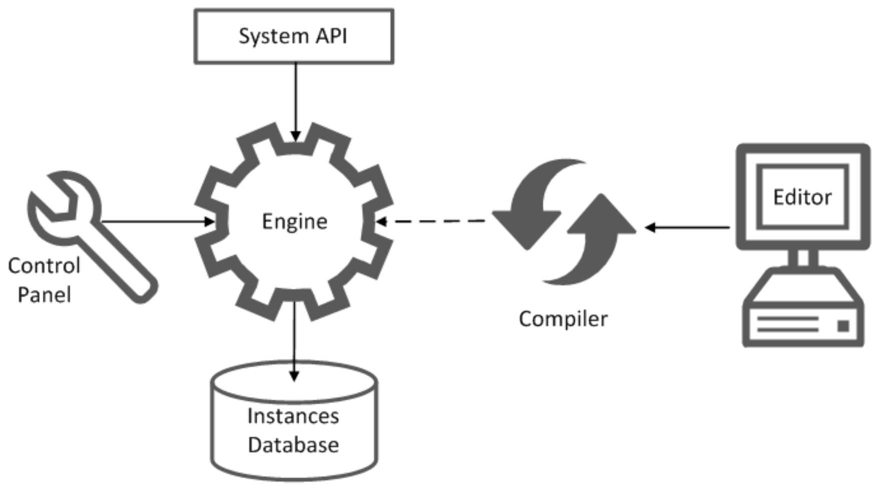

Figure 4.1: The architecture of the collaboration system

Once the configuration and collaboration specifications are written using the collaboration editor (see Section 4.5) and compiled (see Section 4.6) into executable collaboration specifications, they are deployed in the collaboration engine to be executed. When the engine receives an event requesting a collaboration instantiation, the engine creates an instance of the collaboration according to the corresponding collaboration specifications; then, the engine places the newly created collaboration instance in the pool of active collaboration instances (see Section 4.3). The engine continuously listens for incoming events, whether from external systems or from active collaborations. Receiving an event, the engine triggers it on the corresponding collaboration instance which results in the execution of the relevant event-handlers of the instance. If there is an event-handler defined in the collaboration for the incoming event type, and if the associated conditions are satisfied, the engine executes the enclosed actions, which may involve state transition, sending a notification, or any other method call.

The collaboration engine is composed of four main components.

- The **Skeleton** defines the structures of the language elements and the relations among them. The skeleton is in the form of a class library containing the abstractions of the language elements. These abstractions specify the attributes and responsibilities of the elements. For example, the AbstractData, which is the abstraction of the data elements of the collaboration instances, declares the generic fields of collaboration instances, e.g., ID and creator, and defines the "serialization" and "deserialization" of data as the responsibilities of every collaboration data element. The compiler (Section 4.6) follows these abstractions and creates executable specifications to implement them. The structure of the skeleton including the abstractions and their relationships is depicted in Figure 4.2.

- The **Container** is the placeholder for the collaboration instances during their execution. The container manages the poll of running collaboration instances and directs the events to the

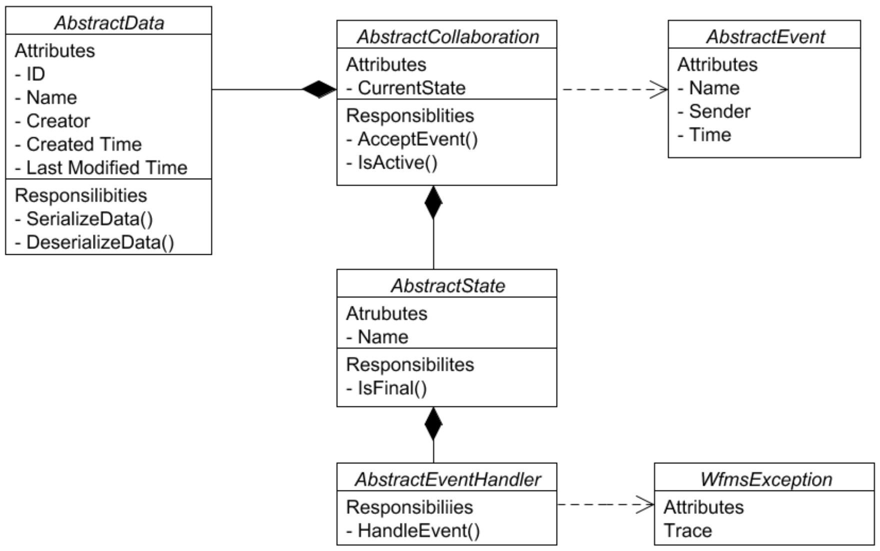

Figure 4.2: The structure of the engine skeleton

relevant collaboration instances. Upon receiving an event, the container retrieves the corresponding collaboration instance and places it the poll; then, the container pass the event and the collaboration instance to the processor in order to initiate the execution process.

- The **Processor** executes events on collaboration instances. It provides the basic execution capabilities of the collaboration engine. The processor supports checking conditions, manipulating fields, invoking web-services, throwing exceptions, instantiating collaboration instances, and all the other execution requirements. In fact, the processor implements the behaviors specified in the collaboration language. Therefore, it is the responsibility of the processor to execute collaboration instances according to their specifications and result in the expected outcomes.

- Finally, the **Stubs** implement the roles, relations, and services defined in the configuration specification. In fact, the stubs are local methods which facilitate invocation of web-services. As opposed to other pieces of the engine which do not require adapting to the context system, the stubs are configured according to the provided configuration specification in the system. This configuration is done automatically by the collaboration system.

The collaboration engine has been implemented as a dynamic web-project in Java. It is deployed as a web-service in Apache Tomcat web server.

## 4.2 The API

Interactions between the collaboration engine and the external systems used by the people involved in the collaboration activity are possible through the system API. Using the system API, external components send events to the collaboration system, which are subsequently directed the corresponding collaboration instances in the engine. It is a web-API which facilitates integration of the collaboration system in the projects based on service-oriented architecture. The API is developed according to REST architectural style in which the communications are stateless and done through HTTP methods, i.e. GET and POST.

The system API exposes collaboration instances as web resources. Therefore, every collaboration is accessible through HTTP method calls using its URI. URIs of collaboration instances have the following form: “[Base Address]/[Type ID]/[Instance ID]” where [Base Address] is the web address of the root of the collaboration system, [Type ID] is the identifier of the type of the collaboration instance, and [Instance ID] is the identifier of the particular collaboration instance. For example, “[Base Address]/Publishing/1” is the URI of an instance of the publishing collaboration whose identifier is wf1.

Events are sent to collaboration instances using POST method calls. Events are also treated as web resources with the following URI form: “[Collaboration Instance URI]/[Event ID]” where [Collaboration Instance URI] is the URL of a collaboration instance as described above and [Event ID] is the identifier of a type of the event. By invoking POST method on a URI in the above mentioned form, an event of that type is triggered on the corresponding collaboration instance. For example, when a POST method is called on “[Base Address]/Publishing/1/Submit”, a Submit event is sent to the specified instance of the publishing collaboration. Besides, the parameters of POST method calls are used for assigning value to the fields of created events. For this purpose, the method calls should carry parameters whose names match the names of the events’ fields; then, the values of the fields will be set to the values of those parameters.

In addition, the system API provides the capability to get information about collaboration instances using GET method calls. This information includes any generic or collaboration-specific data of a collaboration instance such as the state of the collaboration or the value of a field. This capability is particularly useful for automated components which require accessing some collaboration data in their execution processes. Similar to events, every piece of collaboration data is accessible as a web resource through GET method calls. The URI of these resources are in the following form: “[Collaboration Instance URI]/[Data Element]” where [Collaboration Instance URI] is the URL of a collaboration instance and [Data Element] is the name of a data element in the collaboration instance. For example, invoking GET method on “[Base Address]/Publishing/1/State” returns the state of the specified instance of the publishing collaboration.

The REST API is automatically configured by the collaboration system according to the provided configuration and collaboration specifications. Web resources are defined based on these specifications such that [Type ID], [Event ID], and [Data Element] respectively point the name of a specified collaboration, the name of an employed event in a event-handler or entry of the collaboration, and the name of a generic or collaboration-specified field of the collaboration. Upon updating a specification or adding a new one, the system re-configures the API to handle the modifications.

## 4.3 Instances Database

The instances database is responsible for managing and archiving collaboration instances. The instances database maintains the active collaboration instances and also archives completed/terminated ones. This component is composed of a database management system and an access layer on top of it.

The database management system stores the collaboration instances. We employed MySQL as the database in our system. It is possible to replace MySQL with any arbitrary relational database management system, but this task requires some minor modifications.

Every collaboration instance is stored as a record in the database. Each record maintains a set of generic data about the collaboration instance including: the collaboration identifier used to uniquely identify the collaboration instance, the user who created the collaboration instance, the instance's creation date, the instance's last modified date, the type of the collaboration instance, the current state of the collaboration instance. In addition, each record has some collaboration-specific data based on the type of the collaboration instance. This collaboration specific data is actually the most recent values of collaboration instance's data elements, i.e. fields and sub-collaborations.

For performance reasons, we store the active and completed collaboration instances in separated tables which have exactly the same schema. In the longer run, the number of active collaboration instances would be relatively small compared to archived instances; therefore, keeping the active ones in a dedicated table results in faster database operations and also ensures that the response time does not suffer over time as the number of archived instances grows.

The access layer integrates the database with the collaboration engine. It transforms collaboration instances to records in the database and vice versa. The engine uses the access layer to store and retrieve collaboration instances in/from the database. To store a collaboration instance, the layer extracts out the generic data and serializes the collaboration-specific data from the collaboration instance, then combines them into a record in the database. To retrieve a collaboration instance, the layer selects the corresponding record from the active table and builds the collaboration instance accordingly which involves deserialization of the collaboration-specific data.

In addition, the access layer provides a collaboration caching mechanism for enhancing the performance. The collaboration cache helps to decrease the number of database operations by maintaining a dynamic set of collaboration instances. The goal is to cache the collaboration instances which are the most probable ones to be retrieved next. To achieve this goal, the caching mechanism assumes that a new event is more likely to be related to a recently accessed collaboration instance than a random one. Therefore, the cache is basically a data structure which maintains the most recently accessed collaboration instances. The size of the cache can be configured at the engine-deployment time. When the collaboration engine asks the access layer for a particular collaboration instance, the access layer first checks the cache to see if it contains that instance. If the instance is found in the cache, the access layer simply returns the instance without engaging the database; otherwise, a normal retrieval process is performed, which involves fetching the record from the database and transforming it to the collaboration instance. In the latter case, the newly retrieved collaboration instance replaces the member of the cache with the oldest timestamp; while in the former case, only the timestamp of the collaboration instance will be updated.

## 4.4 The Engine Control Panel

The Control Panel was developed to support the administration of the collaboration system. It is a web-based application used mainly by the administrators of the system. Using this component, the administrators can perform the following tasks.

- Starting, stopping and checking the status of the collaboration engine.
- Getting various pieces of information regarding collaboration instances, such as the list of active collaboration instances of a particular collaboration type, the state of a collaboration instance, and the values of any generic or collaboration-specific data element.
- Reviewing the logs of the system such as the incoming events log, the outgoing service calls log, and the system exceptions logs. These logs consist of the descriptions of the incidents in addition to their timestamps and the points in the code where the incident was detected.
- Deploying the compiled configuration and collaboration specifications in the engine. After using the editor and the compiler to compile collaboration specifications, system administrators upload the specifications to the control panel; then, the control panel deploys the compiled specifications in the collaboration engine and makes it ready for instantiation. In addition, the control panel is used when system administrators what to replace a specification with a new one.
- Manually modifying the data elements of collaboration instance. This feature is mainly used for handling exceptional cases or correcting the values of data elements of collaboration instances. For example, a system error may result in damages to the data of a collaboration instance; therefore, it is required to correct the data in order to put the collaboration instance back to action. For this purpose, a system administrator loads and modifies the values of the data elements. Using this feature should be avoided as much as possible, except when it is highly required, since even tiny mistakes in the manual modifications may corrupt the collaboration data or put the collaboration instances in unstable states.

## 4.5 The Editor

The editor is used for creating and modifying collaboration specifications. In fact, it is a customized textual editor configured according to the grammar of our collaboration language. The editor facilitates specifying collaborations by providing text processing.

In order to provide a user friendly interface for collaboration description, we employed Xtext [9], a framework for building textual editors. Xtext is an open-source language development framework that supports the creation of textual domain specific languages. Using Xtext we developed an Eclipse-based environment for collaboration development which offers editing experience similar to Java IDEs. Accordingly, we provide the Xtext framework with the grammar of our Collaboration language; based on the grammar, Xtext generates a parser that reads the textual collaborations and builds the Abstract Syntax Tree (AST) meta-models of the collaborations.

Our Eclipse-based collaboration editor provides syntax coloring, code completion, code folding, a configurable outline view and static error checking for the given collaboration model. All these basic syntactic model processing is possible based on the defined grammar of the collaboration. The syntax highlighting and outline view features are resulted directly from the AST of the model and the proposed grammar rules, and the code completion feature works based on what the generated parser expect to see next in the model.

Note that all these properties are customizable and can be programmed to include further facilitating features such as rename refactoring and hovers. Besides, It should be mentioned that the editor only performs some static error checking using the corresponding AST representation of the collaboration model, and the advanced model validity checking and other model processing are done in the compiling and translating phase.

## 4.6 The Compiler

Since the collaboration model represented in our collaboration language is not understandable by the rest of our Java based framework, the first obvious step after its specification is to transform it to its Java representation. To achieve that, we needed a compiler to go through the collaboration model, check its validity with respect to the syntactic and semantic rules, and translate it to the target language. The separation of the basic syntactical parsing in the editor and the advance semantic processing and translating in the compiler component helps to focus on the user friendliness of the editors on the one hand, and implement more sophisticated logic on the other hand.

All elements of the compiler, from lexical analyzer to target code generator are implemented outside the Xtext framework. Although the Xtext framework used for building the collaboration development editor could also provide basic lexical and syntactic processing, we built our own parser and analyzer as part of the compiler. In order to handle cross reference linking and also provide semantic processing of the model we needed to have the AST model of the collaboration

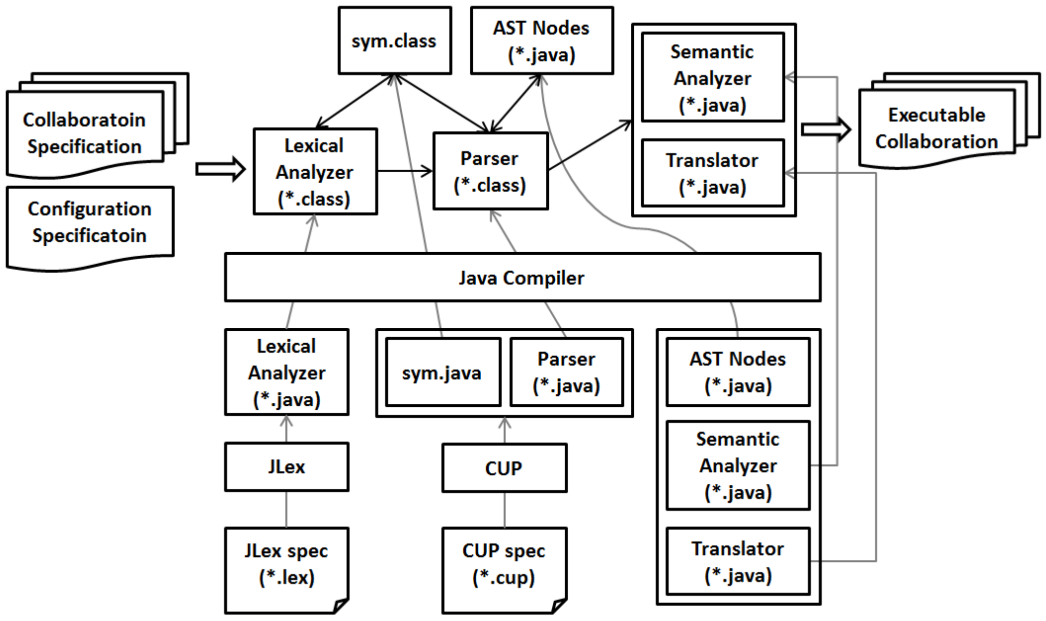

Figure 4.3: The structure of the collaboration compiler

description. We did not make use of the AST created by Xtext since it was built based on the left recursion free grammar and had a complicated structure. The AST created by Xtext had more nodes than necessary and its structure was not easily translatable to the target model. Consequently, we build our own compiler with all fundamental components. The provided implementation might not be the most efficient translator, it could transform the collaboration model to the target code in exactly the way we needed. Figure 4.3 illustrates the high-level structure of our compiler.

The input of the compiler is one configuration specification file, and one or more collaboration specifications; the output of the compiler is the corresponding collaboration implementations in Java format. Whenever the configuration specification is changed, all the collaboration specifications should be recompiled with respect to the updated configuration. Besides, if a collaboration specification is changed only that single collaboration and the ones using it (either inheriting that collaboration or including its instances) should be recompiled.

The components of the compiler include:

A **scanner**, a.k.a. **lexical analyzer** breaks a textual input stream of characters into meaningful tokens of the language (such as keywords, numbers, and special symbols). Since writing lexical analyzers manually can be a tedious process, software tools have been developed to ease this task. JLex, as one of these tools, takes a specially-formatted specification of the target tokens and a series of rules for breaking the input stream into tokens, and creates a Java source file for the corresponding lexical analyzer. Lexical analyzer is the one that decides which tokens(characters and words) has been presented in the textual model, and which ones should be returned to the parser for further syntactic analyzing, and which ones should be ignored, e.g. comments. Therefore, the rules included in the lexical specification (.lex) describes the action the lexer should perform in case on recognizing each of the tokens; for example in case of seeing an string started with “//” the whole line of the syntax should be considered as a comment and no token should be returned to the parser, whereas whenever “;” is recognized SEMI-COLON token should be returned to parser which could indicate the end of a line.

A **parser**, a.k.a. **syntactic analyzer**, is responsible for analyzing the textual model, made of a sequence of tokens, to determine if it is correctly structured based on the given grammar. In order to generate the parser for our compiler, we employed the Java based Constructor of Useful Parsers (CUP). CUP is capable of generating a LALR parser from the grammar specification of the collaboration language. The grammar specification can also include the action the parser should perform in case of seeing each production, e.g. returning the corresponding AST node while realizing a language element. Therefore, the AST of the collaboration specification can be built at the same time the parsing occurs. To achieve that, the needed AST nodes should be defined separately, in Java, and the corresponding package should be included in the specification file (.cup). Running the CUP generator on JVM with respect to the specification, the system will produce two Java source files containing parts of the generated parser: *sym.java* and *parser.java*. The sym class contains the constant declarations for terminal symbols and can be used by the scanner to refer to symbols (e.g. while returning sym.SEMI-COLON for “;”), and the parser class implements the LALR parser and is capable of extracting the AST of a collaboration specification for further model processing as semantic analyzing and translation to the target code .

A **semantic analyzer** is responsible for adding semantic information to the AST and building the symbol table respectively. All type and semantic validity checks should be done in this phase, e.g. all the mandatory event parameters (indicated by “*” while defining the event) of the event should be not null while triggering the event. Therefore, this component makes sure that the specification has been developed correctly, and it is can be translated to a meaningful executable collaboration. Since semantic analyzer needs the parse tree, this phase logically follows the parsing phase, and precedes the code generation phase for translation. The semantic analyzer of our collaboration compiler is also implemented in Java.

A **translator**, a.k.a. **code generator**, goes through the AST and the symbol table created in the previous phases, understands the logic of the model and transforms it into a Java representation of the collaboration which is understandable by the rest of the collaboration engine. Translating the collaboration specified in our collaboration language is the most crucial phase of our compiling procedure. Since the output language of our compiler is a high level programming language (Java) not a low level machine language, the translation does not involve resource and storage decisions, e.g. assigning registers to variables. However, it needs to explore the logic of the collaboration specification, implement event handling procedures for inter collaboration communications, facilitating service calls within a collaboration, managing the state transitions of a collaboration instance,

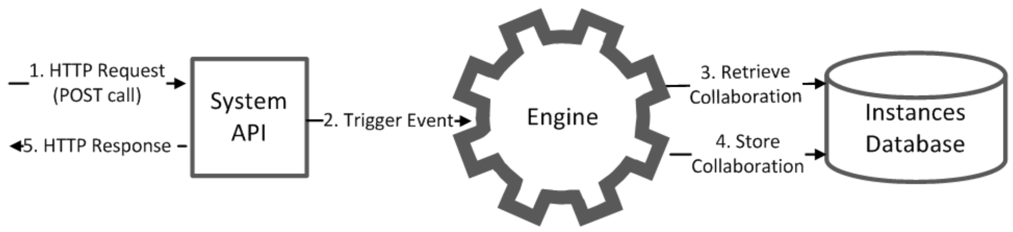

Figure 4.4: The communication diagram of the run-time components

and etc. Each of these aspects needs considering details about how each of the corresponding elements (collaboration instances, events, states, and etc.) will be used further in the collaboration engine. Details about how transformation from the original collaboration model (based on its AST and symbol table) to the target Java files, have been presented in Appendix A.

## 4.7 Interactions

As we introduced the six components of the collaboration system, each of them takes a role in the management of collaborations. However, their levels of engagement are different. Collaboration engine, system API, and instance database are run-time components which are actively involved in the execution of the collaboration instances. They are called into action when a new event is arrived. The collaboration editor and compiler are design tools which are used for specifying new collaborations or modifying current ones. The design tools are not engaged in handling of events and execution of collaborations; therefore, they are used less frequently than run-time components. Finally, control panel is a managerial tool used for administrative tasks. The frequency of its usage depends on the collaborations and the amount of the administrative tasks that should be performed. A communication diagram illustrating the interactions of the run-time components in the course of executing a collaboration instance is provided Figure 4.4.

According to this figure, the interaction is initiated when an external component sends a HTTP request, in the form of a POST call, to the system API. Receiving the request, the API creates an event accordingly and directs in to the collaboration engine. The engine acts upon the event by retrieving the corresponding collaboration instance from the database and executing the event on the instance. The execution may involve various actions performed by the engine such as changing the state of the collaboration instance or calling other services. Finally, the engine stores the collaboration instance back in the database and the system API sends back a response to the initiator indication the completion of the execution.

# Chapter 5

# The Integration Model

One of our main goals in the design the collaboration system was the generality of its applications and its ability to support a variety of resource-centric collaboration projects. In fact, the collaboration system should be integratable to any arbitrary ecosystem of tools, in order to add the coordination support required for their collaborative activities. For example, if we have a collaborative real-time editor which provides a collaboration environment but falls short in supporting the coordination requirements, we can integrate the collaboration system with the editor to enhance the editor with the coordination capabilities of the collaboration system. To facilitate the integration process, we have developed a methodology for integrating the collaboration system within an ecosystem of other systems and tools as depicted in Figure 5.1.

In this section we discuss at a high level the model (and process) of integrating our collaboration system with an existing set of tools that the collaborators use to manipulate and share their resources.

## 5.1 The Collaboration System

The term “collaboration system” stands for the software framework described in Chapter 4. The main responsibility of the system is to coordinate the users’ collaboration at run-time, using the collaboration engine, according to given collaboration specifications. The collaboration system generally refers to all of the components of our collaboration system as depicted in Figure 4.1. However, the collaboration engine is at the center of focus here as it is the main component involved in the execution of collaboration instances.

## 5.2 The Base System

The base system includes the set of tools that the users use for collaborating on resources. It is actually the target component for which we want to provide coordination supports by our collaboration system. Base systems support resource manipulations and collaborations among users. Each base system has its own architecture which can be significantly different from another base system.

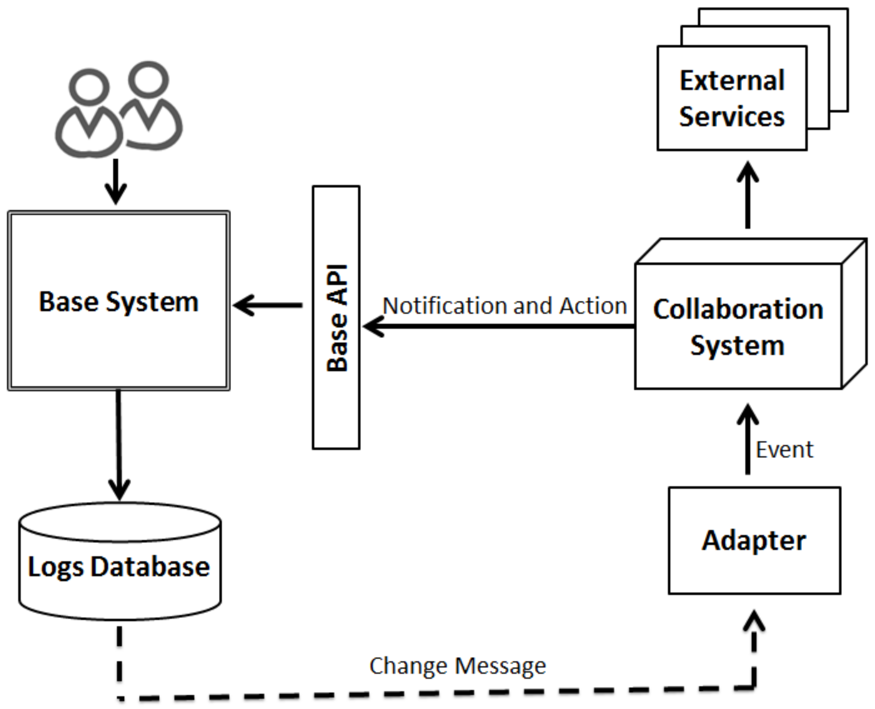

Figure 5.1: Our suggested model for integrating the collaboration system with other tools

However, they usually share variants of the following core components (a base system probably has other components also but they are irrelevant to our system).

- **Editors and Tools** are responsible for manipulating resources. They are used directly or indirectly by users to work on resources. Editors and tools retrieve resources from the repository and provide them to the users; then, users use their facilitation and capabilities to collaborate on resources, and finally, store back the resources in the repository. In addition to resource manipulation, the editors and tools enable the interaction between collaborators, for example using a notification mechanism. In our context, we assume that the editors have user interfaces, and in fact, provide an editing environment for users, while tools are in the form of services which are invoked by editors or other tools. Online text editors and spell checkers are the example of editors and tools respectively.

- **Repositories** are responsible for housing resources. They provide an interface (typically a REST API) for Editors and Tools to access and update resources. A repository can be implemented using a conventional database management system such as MySQL, a document management system such as Fedora, or any other software system capable of maintaining and publishing resources.

In the model, there are a variety of editors and tools working with a repository. For simplicity, we assumed that there is only one central repository; however, it can be any other form of system for housing resources such as a distributed repository, but it is out of scope of our model and has no impact on it. In addition, we do not discuss how to develop a base system but we assume that one already exists and our task is to integrate it with our collaboration system. In other words, we want to orchestrate a base system, composed of a repository and a set of tools, using the collaboration system.

## 5.3 The Base API

The Base API mediates the interaction between the base system and the collaboration system. This API is actually an integrated interface into different elements of the base system. The collaboration system employs the base API to communicate with the base system, or call any editors or tools. Examples of when the base API is used by the collaboration system include: the collaboration system needs to update one resource; or the collaboration system wants to automatically run a service on a resource as one step of a collaboration specification. The base API is composed of a set of methods implemented as REST API. Calling each method results in calls to the editors, tools, or repository of the base system. The methods on the base API can be categorized in two groups.

- **Access Methods** are used by the collaboration system to get information from the base system. The information can be regarding various elements of the base system, such as the state of a particular resource or the profile of a particular user of the base system. This information is usually used in conditional elements of the collaboration specifications. For example, assume that the execution of an action in a collaboration depends on the state of a particular resource; therefore, the engine needs to get the state of the resource in order to decide whether the action should be performed or not. This information can be accessed by one of the access methods of the base API.
- **Management Methods** are used by the collaboration system to update some information or trigger some actions in the base system. As opposed to access methods, which are “safe” methods that only retrieve information and have no side effects, the management methods are used to change the state of the resources in the base system. These effects can be in the forms of changing the content of a resource, updating the profile of a user, sending notification to a user by an editor, etc.

The base system may have an already-available API which can be used as the base API by the collaboration system. However, in many cases the base system does not have such an API, or the API does not contain all the functionalities required by the collaboration system, or it does not support the communication protocol expected by the collaboration engine. In this case, we need to implement such an API as a part of the integration process. The set of methods required for the API depends on capabilities of the base system and the functionalities expected by the collaboration engine in order to execute collaboration instances.

## 5.4 Logs Database

As we employ an event-driven approach for modeling and implementing collaborative activities, the collaboration engine needs to know about the events that occur in the base system. In fact, these events are the triggers that will likely result in execution of some actions according to the collaboration specifications, in the collaboration engine. In our context, the events denote all the actions performed in the base system, e.g., creation or modification of a resource, changes on the profiles of users, or invocation of a tool. Therefore, we need a mechanism to inform the collaboration engine about them. The Log database is the component which facilitates this communication.

The logs database maintains the logs of actions of interest in the base system. The log database is composed of a data repository software system, typically a conventional database management system for housing the action logs and an interface for registering the logs. The base system is responsible for populating this database with action logs; therefore, whenever an activity of interest is performed, the base system should generate and place the corresponding logs into the database through its interface.

In addition, the logs database is responsible for informing the collaboration engine about the performed actions in the system. For this purpose, the logs database employs a publish-subscribe mechanism in which the components, which are willing to get informed about the actions logs, register themselves with the logs database; whenever a new action is logged, the logs database informs all the registered components about the action. Most data repository software systems have in-built or complimentary mechanisms for propagating the changes messages, e.g. triggers in MySQL; therefore, the logs database can employ this mechanism for publishing action logs.

A side benefit of the logs database is that the logs kept in it can be used as the indicators of the history of the base system. Therefore, in the case of an error, the administrator of the base system can use the logs to identify the sources of the errors. In addition, the logs can be used for statistical analysis on the actions performed in the system; for example, to recognize the usage patterns of the system.

## 5.5 The Adapter

The change messages generated by the logs database contain the information required by the collaboration engine in order to execute collaboration instances. However, the REST API of the collaboration system may not directly understand the change messages; it expects the information in the form of events communicated to the engine though HTTP method calls. The events should conform to the specific formats defined by the collaboration system.

The Adapter is responsible for transforming the actions logs into events understandable by the collaboration engine. Effectively, this is a middleware between the base system and the collaboration system. The adapter registers itself with the logs database and listens to the change messages publish by the logs database. Whenever a change message is received, the adapter creates an event based on the action log and sends it to the collaboration engine.

Basically, the adapter is composed of (a) a parser to tokenize the incoming change messages, (b) a converter to create events in the format expected by the engine, and (c) a transmitter to send the events to the collaboration engine. Based on the publishing mechanism used in the logs database and the format of the change messages, the adapter should be modified and configured to enable the transformation, specifically the parser and the converter modules of the adapter. It is more straightforward to implement the adapter as a web-service, but it can be implement in other ways too, e.g., as a script.

## 5.6 External Services

External services are used to extend the capabilities of the collaboration engine. External services can be called during the collaboration-instance execution, using the service call function provided in the collaboration language. Therefore, if some more complex processing during the collaboration execution is required, which cannot be supported by the collaboration language, it can be implemented as a web service using any scripting or programming language, and then be invoked by the engine. In addition, using external services is the mechanism for reusing already-available web services.

## 5.7 Interactions

A typical scenario of interactions among the components of our model is described below and depicted in Figure 5.2. In this scenario we assumed that the engine is provided with configuration and collaboration specifications, and the components of the system are configured according to the integration model illustrated in Figure 5.1.

1. Using an editor, a user of the system requests to access a shared resource. We assume that the resource is already-created and placed in the repository of the base system and the editor is one of the editors and tools of the base system.

2. The editor retrieves the resource from the repository of the base system and presents it to the user.

3. The user works on the resource. At the end of the session, the user requests to update the resource.

4. The editor updates the resource in the repository. In addition, the editor adds action logs on the logs database according to the performed actions by the user.

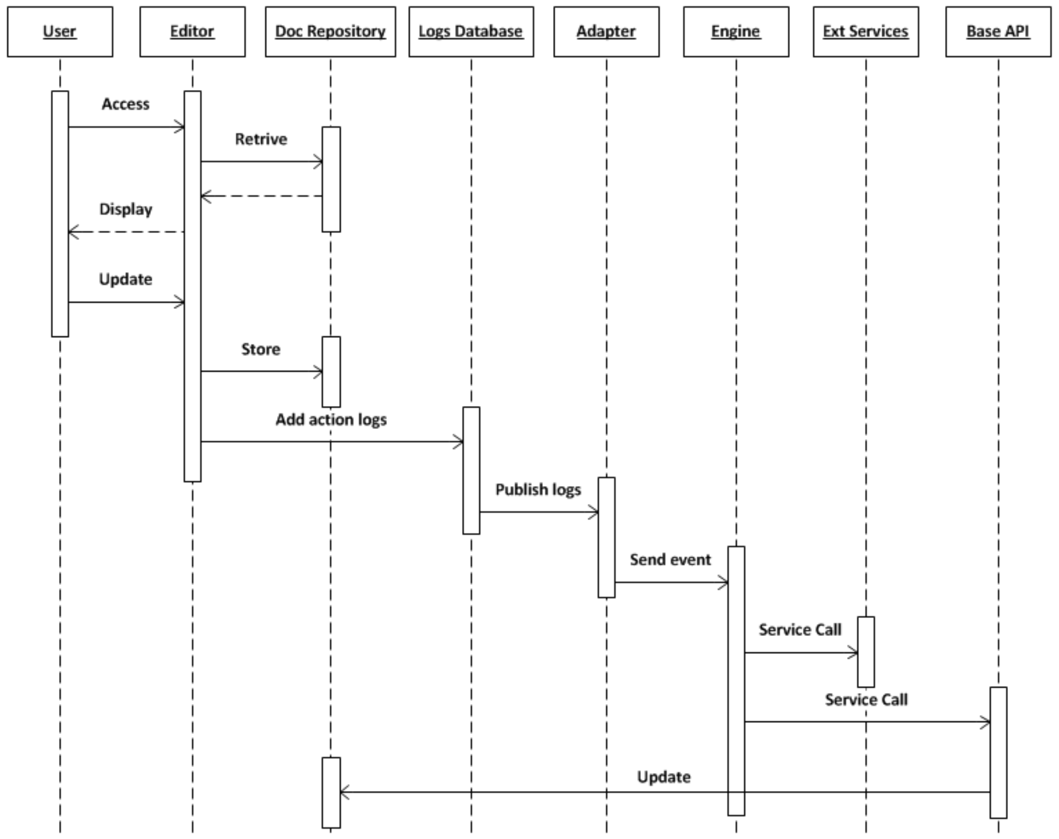

Figure 5.2: A sample scenario of the interactions among components of the integration model

5. Upon addition of new action logs, the logs database publishes change messages regarding the action logs.

6. The adapter receives the change messages; consequently, it creates the corresponding events and sends them to the collaboration engine.

7. The collaboration engine reacts to the events by triggering the events on the related collaboration instances which result in execution of some event-handlers. The execution may involve:

   - internal changes in the state and data of some collaboration instances,
   - calls to some external services, and
   - calls to the base API.

8. Calls to the base API result in invocation of some services on the base system such as update of the shared resource.

While designing the integration model, our primary objectives were ease of integration and low coupling between the collaboration system and the base system. Accordingly, the base system is not aware of the collaboration system. The base system only logs the activities and requests, and it is the collaboration system which listens to the logs with help of the adapter and acts upon. Therefore, it is possible to change or replace the collaboration system completely without any need to modify the base system. In addition, the adapter makes the integration easy since we can change the base system without any need to modify the engine; but we only need to modify the adapter. Similarly, the collaboration system is only coupled to the base API; therefore, as long as the API is not changed, it is not important how each method of the API is implemented or if the internal structure of the base system changes, e.g. a tool is replaced or a new editor is added.

We believe that our integration model works for the majority of resource-centric collaboration projects. Although this model is not the only way that the collaboration system can be employed and integrated with a base system (and one can conceive of conflicts between the integration methodology and the base system's restrictions to information access) this method and corresponding tools guide the process of integrating our collaboration system with existing interactive systems to coordinate the activities of their users.

# Chapter 6

# Case Studies

In the previous chapters, we discusses our collaboration language and the corresponding framework. In this chapter, we describe two resource-centric collaboration projects in which our collaboration system has been employed. These cases demonstrate our collaboration system in action and are evidences of applicability of our collaboration system in real-world projects. For both of the projects, we employed and customized our integration model.

## 6.1 The GRAND Project

The GRAND (Graphics Animation and New Media) Network of Centres of Excellence [5] is a multidisciplinary research network exploring the application and advancement of graphics, animation and new media in Canada. The GRAND forum is a web-based application, designed to collect information about the people in the GRAND community, their relations, and their products and activities. Besides, the forum supports delivery of information within the GRAND networks. The GRAND forum has been implemented based on Mediawiki [7] framework which is a web-base wiki software application used for building various wikis such as Wikipedia. According to the web site, “the purpose of the GRAND forum is to provide a central repository for up to date information about (a) the GRAND community and activities, and (b) the events of interest to the GRAND community. In addition, the GRAND forum provides the means to collect and disseminate content as necessary for the collaborative and reporting activities of the network, and analysis tools through which to examine the evolution of the network community.”

The main entities of the GRAND network and their relations are depicted in the entity-relationship diagram depicted in Figure 6.1. In this diagram, many of the details are omitted as our intention was just to show a high-level picture of the network. According to the diagram, there are many researchers defined in network who are in fact the users of the GRAND forum. Every researcher in the GRAND network has a role, one of Manager, CNI, PNI, and HQP. There is a Supervise relation among some of the researchers; more specifically, every researcher with role HQP is associated with supervisors who are either CNIs or PNIs. Many projects are also defined in the network. Every

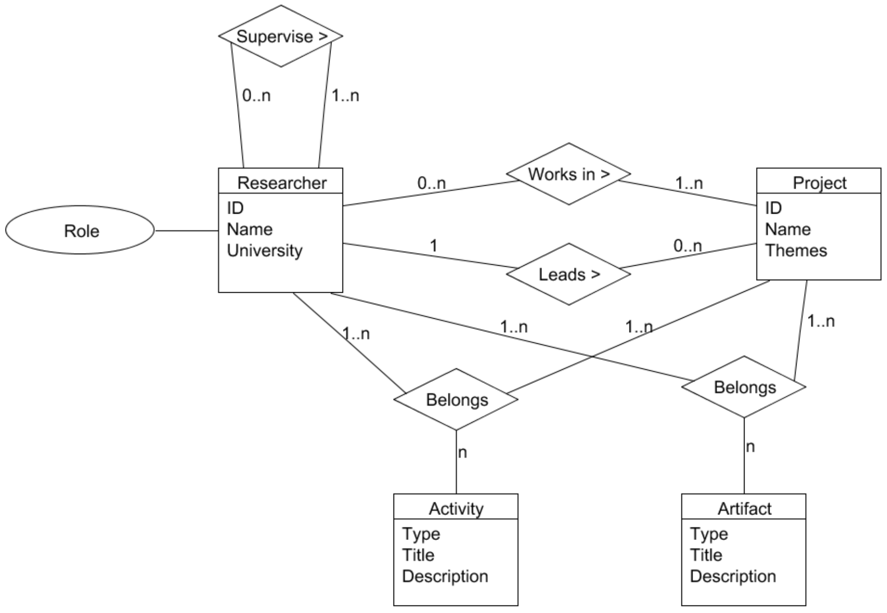

Figure 6.1: The entity-relationship diagram of the GRAND network

project is associated with several researchers, who are the members of the project, and one or two researchers who are the leader and co-leader of the project. In addition, artifacts and activities are among other important entities of the GRAND network. Every artifact or activity is associated with (potentially several) researcher(s) and project(s).

As well as presenting the information about GRAND network, the forum provides the users with the capabilities of updating the network information, such as, for example, adding a new researcher, or deleting an artifact. In addition, the forum has an in-built notification dashboard for every user in which the recent updates relevant to the user are posted; for example, when the administrator modifies the role of a user, a notification will be added to the notification dashboard of the user informing the role change.

In the GRAND forum, it is possible to modify the underlying information resources using the provided HTML forms; however, this does not support the desired flexibility and the dynamics in performing the tasks. In fact, many of the GRAND collaborations require interactions among various researchers; since the forum was not designed to support these interactions, the researchers have to perform the interaction though other channels rather than the forum. For example, when a new researcher requests to join a set of projects, the manager of the system and the leaders of the projects should consent; then, the addition can be done. However, there is not possible to support this collaboration completely in the forum as there is no way to ask the questions and act according to the responses.

Therefore, there is a need for further development in the GRAND forum in order to support more complex and interactive collaborations. While Mediawiki, as the infrastructure of the forum,

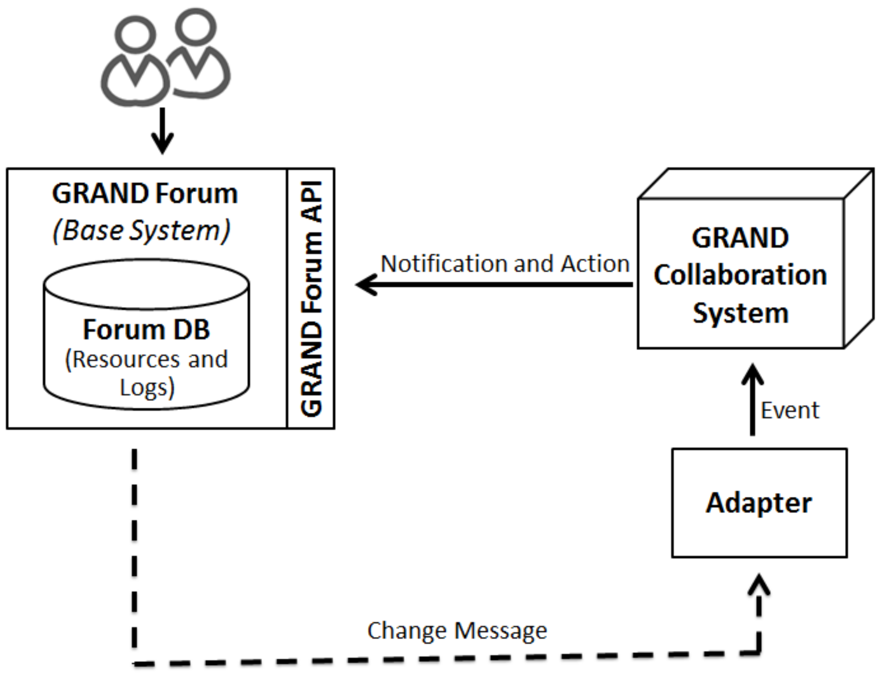

Figure 6.2: The realized integration model for GRAND project

supports addition of new features through extensions, it requires too much effort to implement all of the GRAND collaborations as extensions. In addition, collaborations are typically subject to continuous changes; therefore, hard-coding the collaboration logics into Mediawiki extensions makes the change process extremely difficult.

We first started by exploring the GRAND collaborations in the GRAND forum and identifying their requirements. We came up with a set of basic collaborations for managing the users and their relationships such as collaborations for creating new users and changing role or projects of the users. After analyzing these collaborations, we found out that they share most of their properties and requirements with resource-centric collaborations. For example, the GRAND collaborations are semi-structured and their activities are mostly driven by human participants. Therefore, we decided that our collaboration system can fit appropriately in the GRAND forum and provide the support for implementing the required collaborations. In order to employ the collaboration system in the GRAND forum, we tried to realize the integration model for the GRAND forum. The resulted integration model is depicted in Figure 6.2.

According to this figure, the GRAND forum is actually the base system to which we add collaboration support. In this context, the front-end of the GRAND forum acts as the editors and tools of the integration model. More specifically the HTML forms work as the editors of the base system and the services provided by the Mediawiki framework can be considered the tools. The GRAND forum has an underlying database management system integrated with Mediawiki. It is a MySQL database which acts as the document repository of our model. This database maintains all the data of the GRAND forum. The users of the forum can use the provided forms and pages of the forum to access and manipulate the data kept in the database.

The MySQL database of the GRAND forum also serves as the logs database of the integration model. To realize the logs database using this MySQL database, we created a special table, named Event table, in this MySQL database. The Event table is used for storing the actions performed in the front-end of the forum. In fact, the records of the Event table are the logs of the performed actions. Every record in this table has several fields including: the type of the action log, the creator of the action log, the creation timestamp, the identifier of relevant collaboration instance, and some additional data specific to the type of the action log. The front-end of the forum is responsible for populating this table; therefore, we modified the front-end of the GRAND forum so that when a user performs a collaboration-related action, typically submits a form, the front-end generates the corresponding action logs. For example, when an authorized user fills out and submits the form for creation of a new user account, the forum creates a new action log containing the data of the request and adds it to the Event table.

For publishing the actions logs, we employed the trigger mechanism of the MySQL database. Therefore, whenever a new record is added to the Event table, the defined trigger gets executed; consequently, the action log is serialized as a JSON message and sent to the adapter. We developed the adapter of the integration model a web-service using Java Servlet technology. The adapter was configured based on the format of the JSON messages corresponding to action logs. Receiving a JSON message, the adapter parses the message and creates the events for the collaboration system accordingly. Then, the adapter sends the event to the REST API of the collaboration system. In practice, the adapter is responsible for transforming action logs form JSON messages created by the forum into URL-encoded events understandable by the collaboration engine. The collaboration engine reacts to these events by executing the relevant collaboration instances.

Mediawiki, as the underlying framework of the GRAND forum, facilitates the implementation of a REST API on the forum. It creates the skeleton of the API and provides a mechanism for adding methods to it. Using this mechanism, we developed the REST API of the GRAND forum which in fact is the realization of the Base API of the integration model. This API contains methods for accessing and updating entities and the relations of the forum; examples of these methods include methods for getting the information of a researcher, creating a new project, and adding a membership relation between a researcher and a project.

During the execution of GRAND collaborations, there are situations where some information should be provided to a user or a group of users; for example, when a particular collaboration is completed, all the interested users should be informed about it. Similarly, there are situations where data should be obtained from a user or a group of users; for example, the supervisor of a HQP should provide her decision on approving or rejecting the graduation request of the user before the collaboration can be completed. Therefore, we needed a mechanism in the collaboration system which enables providing information to and asking questions from users.

The GRAND forum already had a notification mechanism. Every user of the forum has a dedicated notification Inbox which can be access through dashboard of the forum. Every notification has a number of fields including the sender, the subject, the description, and the creation time of the notification. This notification mechanism had been mainly used for providing information to users. In order to enable the collaborations to use this mechanism, we added a method on the REST API of the forum by which the collaboration engine can send notifications to users.

Although the notification mechanism enables the collaboration engine to provide information to users, the forum did not provide the collaboration engine with the capability to dynamically receive data from users. To solve this problem, we decided to extend the notification mechanism of the forum. To that end, we developed special types of notifications which ask recipients for data, typically providing some options to the users to choose from. The basic type is ConfirmationNotification, which asks a yes-or-no question from a user. We added methods on the REST API of the forum so that the collaboration engine can create and send notifications of these types to users. Special notifications are also displayed in the notification Inboxes of users. In addition to the fields of the regular notifications, special notifications have additional elements, typically input fields and buttons, depending on the specific type of the notification. Using these additional elements, users can respond to the questions. When a user responds to a special notification, for example, selecting "yes" or "no" on a ConfirmationNotification, an action log is added to the Event table. The added action log contains information about the related question, the type of the response, and the type-specific details of the response. In fact, responding to a special notification will result in executing of an event on the corresponding collaboration instance. The usage of each type of special notifications is not limited only to one specific type of collaboration, but it can be used by any collaboration that requires to get the same set of data from user. For example, any yes-or-no question can be asked from a user using the ConfirmationNotification type. However, if other kinds of questions are needed which can not be supported by already-developed special notifications, new types of special notification should be implemented in the GRAND forum and added to the REST API of the forum.

After developing all the required pieces and completing the integration our collaboration system with the GRAND forum, we worked on implementing a set of GRAND collaborations. For this purpose, we first started by documenting the collaborations, and then specifying them using our collaboration language. As an example, we present one of the implement collaborations, named CNI/PNI User Account Creation collaboration, in the following. The collaboration is responsible for creating a CNI or PNI user account for a new user. The description of this collaboration is provided below.

**CNI/PNI User Account Creation**

- **Description**: This collaboration is used to create a CNI or PNI user account for a new user
- **Initiator**: Any manager or CNI/PNI
- **Main Flow**:
  1. The collaboration is instantiated when an initiator requests for creation of a CNI/PNI user account for a new user. He provides the new user information including the name and the email address of the user. In addition, the initiator should specify whether the new user will be a CNI/PNI. The initiator also provides the proposed projects for the new user.
  2. If the initiator of the collaboration is not a manager then
     (a) Managers get notified that a CNI/PNI user account creation collaboration is instantiated and needs their approval.
     (b) A manager accepts the creation of a CNI/PNI user account for the new user with the provided information.
  3. A CNI/PNI user account is created for the new user.
  4. The new user, managers, and the initiator get notified about the creation of the new CNI/PNI user account.
  5. For each proposed project provided for the user
     (a) If the leader is not the initiator of the collaboration then
        i. The leader of the project gets notified that a new user is proposed as a CNI/PNI for their project.
        ii. The leader accepts the new user.
     (b) The user is added as a CNI/PNI to the project.
     (c) The user, managers, the initiator, and the leader of the project get notified.
  6. The collaboration gets terminated.
- **Alternative Flow 1**:
  1. The alternative flow begins after the step 2.a if a manager requests the termination of the collaboration.
  2. The initiator and admins get notified about it.
  3. The collaboration gets terminated and no account will be created.
- **Alternative Flow 2**:
  1. The alternative flow begins after the step 5.a.i if the leader does not accept the new user.
  2. The user, managers, the initiator, and the leader of the project get notified about it.
  3. Continue the loop from the step 5.

In summary, the collaboration is initiated whenever an authorized user requests creation of a new user account, indicating its role as CNI or PNI. The initiator also associates the new user with one (or more) project(s). First, a manager should decide on approving or denying the request; if the manager approves the request, the new user account is created. Then, the project leaders of the proposed projects should decided on accepting or rejecting the addition of the new user to their projects; if a project leader accepts the addition, the new user is added to the corresponding project.

In order to implement the described collaboration in our system, we first prepared a configuration specification and provided it to the system. The configuration specification reflects the structure of the GRAND resources, the API of the GRAND forum, and the collaborations to be implemented. The specification is provided in Listing 6.1. In the presented specification, we only included the parts which are relevant to our collaboration example, and the remaining parts are omitted for simplification purpose.

```
// Event Definitions
Event Initiate (String username*, String realname*, String email*, String role*,
    Strings pids*);
Event Approve ();
Event Deny ();
Event Accept (String pid*);
Event Reject(String pid*);
...

// Role Definitions
Role CNI (user) : "http://.../?action=api. isCNI" , http://.../?action=api.getCNIs
    ";
Role PNI (user) : "http://.../?action=api. isPNI" , http://.../?action=api.getPNIs
    ";
Role Manager (user) : "http://.../?action=api. isManager" , http://.../?action=api
    .getManagers";
...

// Relation Definitions
Relation Leads(User user, String pid) : "http://.../?action=api. isLeader", "http
    ://.../?action=api.getLeader";
...

// Service Definitions
User POST CreatUser(String wpName, String wpRealName, String wpUserType, String
    wpEmail) :  "http://.../?action=api.addUserAccount";
String POST AddProjectMember(String name, String project) : "http://.../?action=
    api. addProjectMember";
String POST SendNotification(User receiver, String title, String message) :   "
    http://.../?action=api.addGenericNotification";
String POST AskQuestion(User receiver, String title, String message, String
    wf_instance, String accept, String reject) : "http://.../?action=api.
    addConfirmNotification";
 ...
```

Listing 6.1: The configuration specification for the GRAND project

According to Listing 6.1, five events are defined: Initiate event is used for instantiation a collaboration of this type, approve and deny are used by managers for deciding on creation of a new user account, and accept and reject are used by project leaders for deciding on the addition of the new user to their projects. While there are many roles in the GRAND network, here we only included three roles, i.e. CNI, PNI, and Manager. Similarly, the Leads relation is the only presented relation. Finally, services for creating a user, adding a user to a project, send a notification to a user and asking a question from a user are also defined in the configuration specification.

The specification of the CNI/PNI User Account Creation collaboration written in our collaboration language is provided in Listing 6.2. We used the state-based style for writing the specification since the steps of the collaboration are ordered, and the behavior of the collaboration highly depends on its state. The collaboration specification has four distinct states: Waiting, Pending, Terminated, and Completed. The Waiting state represents the situation in which the collaboration waits until a manager decides on either approving or denying the request of a new user account creation. Similarly, the collaboration stays in Pending state until it receivers the decisions of all the relevant project leaders on either approving or rejecting the addition of the new user to their projects. Both the Terminated state and Completed states are Final; however, the Completed state represents the situation where the collaboration is finished and a new user account was successfully created, while the Terminated state represents the situation where a manager denied the request.

```
Collaboration StateBased NiUserCreation {
  // Field Declarations
  String username;
  String realname;
  Users email;
  String role;
  Strings pids;
  String replies;  // this field contains the ids of the projects which are
      accepted or rejected
  String user;

  //  Entry Specifications
  Entry Initiate [CNI, PNI, Manager]{
    username = e.username;
    realname = e.realname;
    email = e.email;
    role  = e.role;
    pids = e.pids;
    If ( role != "CNI" And  role != "PNI"){
      Exception ("The role should be either CNI or PNI");
    }
    Foreach(pid in pids){
      If ( WfCreator Leads pid ){
        replies = replies + pid;
      }
    }
    If ( WfCreator Is Admin ){
      user = CreatUser(username, realname, role, email);
      SendNotification(WfCreator, "Ni User Creation Collaboration", "...");
      Foreach (mng in (All Manager)){
        SendNotification(mng, "Ni User Creation Collaboration", "...");
      }
      Foreach (pid in replies){
        AddProjectMember(username, project);
      }
      Boolean completed;
      completed = True;
      Foreach (pid in pids) {
        If ( ! (replies Contains pid)) {
          completed = False;
          Users leaders = Find ( ? Leads pid);
          Foreach (leader in leaders){
            AskQuestion(leader, "Ni User Creation Collaboration", "...", WfId, "
                Accept "+ pid, "Reject" + pid);
          }
        }
      }
      If(completed){
        To(Completed);
      } Else {
        To(Pending);
      }
    }
    Else {
      Foreach ( mng in (All Manager)) {
        AskQuestion(mng, "Ni User Creation Collaboration", "...", WfId,  "Approve
            ", "Deny");
      }
      To(Waiting);
    }
  }

  // State Specifications
  State Waiting {
    @Approve [Admin]{
      user = CreatUser(username, realname, role, email);
      SendNotification(WfCreator, "Ni User Creation Collaboration", "...");
      Foreach (mng in (All Manager)){
        SendNotification(mng, "Ni User Creation Collaboration", "...");
      }
      Foreach (pid in replies){
        AddProjectMember(username, project);
      }
      Boolean completed;
      completed = True;
      Foreach (pid in pids) {
        If ( ! (replies Contains pid) ) {
          completed = False;
          Users leaders = Find ( ? Leads pid);
          Foreach (leader in leaders){
            AskQuestion(leader, "Ni User Creation Collaboration", "...", WfId, "
                Accept"+ pid, "Reject" + pid);
          }
        }
      }
      If(completed){
        To(Completed);
      } Else {
        To(Pending);
      }
    }
    @Deny [Admin]{
      SendNotification(WfCreator, "Ni User Creation Collaboration", "...");
      Foreach (mng in (All Manager)){
        SendNotification(mng, "Ni User Creation Collaboration", "...");
      }
      To(Terminated);
    }
  }
  State Pending {
    @Accept[CNI, PNI] {
      If( !( e.Sender Leads pid ){
        Exception ("The sender should be the leader of the project.");
      }
      AddProjectMember(user, pid);
      replies = replies + e.pid;
      SendNotification(WfCreator, "Ni User Creation Collaboration", "...");
      SendNotification(e.Sender, "Ni User Creation Collaboration", "...");
      SendNotification(user, "Ni User Creation Collaboration", "...");
      Foreach (mng in (All Manager)){
        SendNotification(mng, "Ni User Creation Collaboration", "...");
      }
      Boolean completed;
      completed = True;
      Foreach (pid in pids) {
        If ( ! (replies Contains pid)) {
          completed = False;
        }
      }
      If(completed){
        To(Completed);
      } Else {
        To(Pending);
      }
    }
    @Reject[CNI, PNI] {
      If( !( e.Sender Leads pid)){
        Exception ("The sender should be the leader of the project.");
      }
      replies = replies + e.pid;
      SendNotification(WfCreator, "Ni User Creation Collaboration", "...");
      SendNotification(e.Sender, "Ni User Creation Collaboration", "...");
      SendNotification(user, "Ni User Creation Collaboration", "...");
      Foreach (mng in (All Manager)){
        SendNotification(mng, "Ni User Creation Collaboration", "...");
      }
      Boolean completed;
      completed = True;
      Foreach (pid in pid) {
        If ( ! (replies Contains pid)) {
          completed = False;
        }
      }
      If(completed){
        To(Completed);
      } Else {
        To(Pending);
      }
    }
  }

  Final State Terminated;

  Final State Completed;
}
```

Listing 6.2: The specification for a sample GRAND collaboration

## 6.2 The CWRC Project

The Canadian Writing Research Collaboratory (CWRC) [2] is a project which aims to create an infrastructure for literary research in and about Canada, in large-scale, cross-disciplinary collaborative models. For achieving this goal, CWRC is developing a web-based service-oriented system. The CWRC system consists of two mains components.

- The **CWRC repository** houses digitized and digital-born materials, and keeps various metadata about these digital materials, such as annotations and cross references.
- A set of **Tools** envisioned to enable collaborative literary studies, by providing capabilities for online writing, editing, annotating, analyzing, knowledge mining, and visualization.

In the CWRC system, the CWRC repository is the central component to which all the tools are integrated. All the documents are kept in the repository which is shared among the CWRC tools. The tools interact to the repository in order to retrieve and store the documents. The CWRC tools constitute the front-end of the system which are used by literary researchers.

The CWRC repository is based on the Fedora [3] repository. Fedora is a general-purpose, open-source digital object repository system. The main functionality of Fedora is its ability to store digital objects and metadata about the objects. A digital object is mainly composed of a number of data-streams which are the actual place-holders for the data and meta-data. Every document in the CWRC system is a stored as a digital object in the repository. In fact, a data-stream is dedicated for storing the content of the document which can have various formats such as text, image, video, or a combination of other formats; and several other data-streams are used for keeping different types of meta-data about the document.

As the CWRC system aims to deliver a collaborative working platform, it needs to support more than just the basic viewing and editing capabilities provided by the tools integrated with the repository. In fact, there are many collaborations relevant in the context of literary research, such co-authoring an article or peer-reviewing a scholarly publication. These collaborations typically require collaboration of some researchers in order to produce the final result. The CWRC system wants to support these types of collaborations; however, the combination of the repository and tools cannot provide the required support by themselves. Therefore, there is a need for a component which enables and coordinates the collaborations among researchers in the CWRC system.

In order to add the required collaboration supports to the CWRC system, we started by exploring the user stories of the CWRC system. The list of user stories, which had been previously complied by the CWRC team, included the information about the job functions of various users and the interactions among them. We found out that most of the CWRC collaborations are formed around centered documents. In fact, the CWRC collaborations are focused on managing different types of documents through their life-cycles. In addition, the CWRC collaborations are composed of loosely-ordered steps which require simple types of coordination. Having all these properties, the CWRC collaborations can be considered as genuine cases of document-centric collaborations. Therefore, we concluded that our collaboration system can appropriately provide the required support for the CWRC collaborations; consequently, we decided to employ our collaboration system in the CWRC system.

In order to integrate the collaboration system with the CWRC system, we started by preparing the integration model according the components of the CWRC system and our generic integration model. The resulted integration model is depicted in Figure 6.3.

According to this figure, the CWRC repository is the base system to which we want to add the collaboration support. In the CWRC system, the CWRC repository acts as the repository of the integration model and the CWRC tools work as the editors and tools of the model. The CWRC tools consist of a group of in-house developed editors and tools, such as CWRC Writer by which the users can edit and annotate scholarly texts. The set of CWRC tools are not closed, but new tools may be developed and added to the system. The users of the system employ the provided tools to work on CWRC documents.

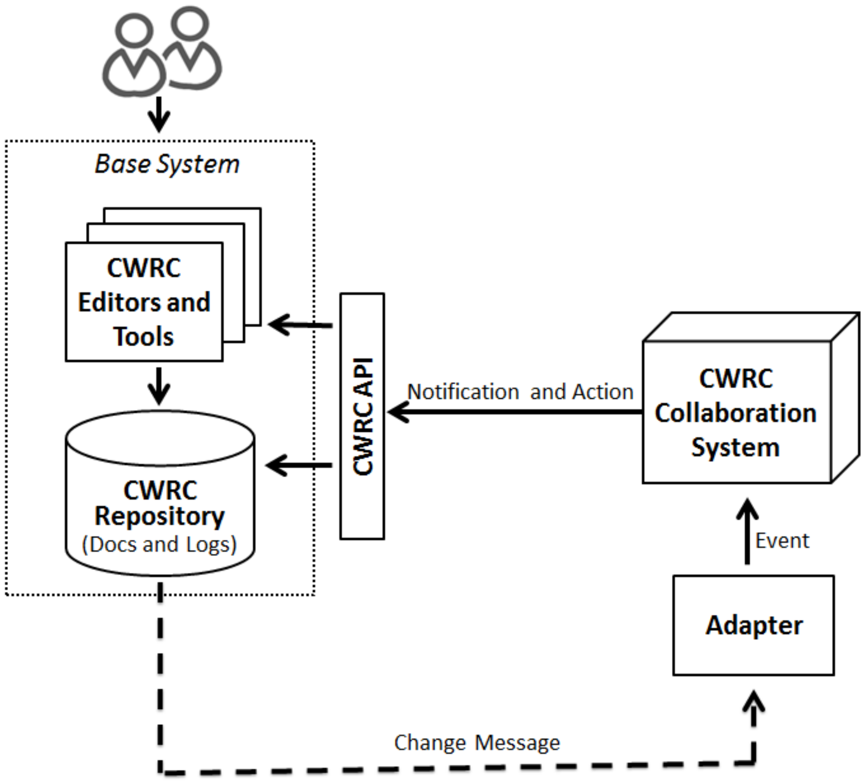

Figure 6.3: The realized integration model for the CWRC project

The log database of the integration model is also implemented using the CWRC repository. We allocated a data-stream of every digital object to collaboration-related metadata. This data stream, which is named workflow metadata data-stream, maintains the logs of actions performed on the corresponding document. In fact, the content of a document and the action logs related to the document are kept next to each others in two data-streams of the corresponding digital object placed in the CWRC repository. The action logs, also known as collaboration metadata stamps in CWRC system, contain information about the actions which users have performed on document using tools, for example adding a new section to a document or correcting the grammatical errors.

The information needed for creation of an action log is provided either automatically by the CWRC tool or manually by the user. It is preferred that CWRC tools create action logs; however, there are situations where the tools cannot exactly know what the users have done; therefore, it becomes necessary that the users provide the required information. For handling both cases, an API and a user-interface for creation of new action logs were developed. If a tool has all the required information, it automatically creates the action log using the API; otherwise, the user-interface is displayed and asks the user to provide the required information.

Every action log, also known as workflow metadata stamp in CWRC system, is composed of several attributes including the ID of the user who performed the action, the ID of the tool which created the log, the time and date of creation of the action log, the type of the performed action, a status indicating the completion level of the action, and some additional information about the action. The actions logs are stored as XML documents into workflow metadata streams. In addition to collaboration-related purposes, the stored action logs of a document provide valuable information about the history of the document.

Fedora, as the underlying implementation of the CWRC repository, provides a messaging mechanism based on Apache ActiveMQ [1], an open-source message broker. The Fedora messaging mechanism enables sending change messages to interested parties whenever digital objects in Fedora repository are modified. Every change message contains the new value of the modified digital object in addition to some additional information about the modification, such as the modification date. For publishing action logs, we first tried to employ the Fedora messaging mechanism. However, this messaging mechanism did not provide us the flexibility we required. For example, we were interested to get the most recent action log added to a modified collaboration metadata data-streams, but the messaging mechanism could only send the whole content of the data-stream. Therefore, we had to find another way by which the adapter gets notified about the creation of new action logs in the CWRC repository. To solve this problem, we decided to implement the messaging ourselves, so some modifications were made to the action logs creation API. As a result, whenever the action logs creation API is called, besides adding an action log to the corresponding collaboration meta-data stream, it also informs the adapter about the action log directly.

The adapter was configured according to the XML schema designed for storing action logs in the CWRC repository. Receiving a message containing an action log in XML format, the adapter parses the message and creates the events for the collaboration system accordingly; then, the adapter sends the event to the REST API of the collaboration system. In practice, the adapter is responsible for transforming action logs from XML format as they are stored in the CWRC repository into URL-encoded events understandable by the collaboration engine. The collaboration engine reacts to these events by executing the relevant collaboration instances.

During the execution of CWRC collaborations, there are situations where some information should be provided to a user or a group of users; for example, a researcher should get informed whenever a document is assigned to her. Similarly, there are situations where some data should be obtained from a user or a group of users; for example, an editor should provide her decision on approving or rejecting a submitted article. Therefore, the CWRC system needed a mechanism which enables providing information to and asking questions from users. Therefore, we developed a notification management component for the CWRC system.

The notification management component mainly consists of a database responsible for housing notification, an API by which notifications can be added to the database, and a front-end responsible for displaying the notifications to users. Every notification is composed of several fields including the creator, the recipient, the subject, the description, the corresponding document, and the creation time of the notification. In addition, every notification can have a number of response options.

The options specify the possible responses that the user can provide to the notification. In fact, by adding response options to notification, questions can be asked from users and their answers can be obtained. For example, if a collaboration requires obtaining the decision of an editor about a submitted article, a notification in regards of this matter is sent to the editor; besides the general information about the question, two response options are provided to the user, i.e. Accept option and Reject option. For collecting the responses to a notification, each response option of the notification includes a URL; the URL points the service which is invoked when the corresponding response option is selected be the user.

The notification front-end was integrated into the main-user interface of the CWRC system. Therefore, when a user logs-in into the CWRC system, the notifications are displayed to the user. For the notification which requires an answer from the user, the options are also displayed to the user next to the notification; upon clicking on one of the response option buttons, the service corresponding to the selected option will be invoked.

Having all these pieces developed and integrated, we specified a few sample CWRC collaborations and deployed them on the CWRC collaboration engine. In this way, we could examine the integration of the CWRC collaboration system with the CWRC tools and repository. However, the CWRC system is under development and some services required by the collaboration system are still missing. Specifically, an API into CWRC system through which collaborations can perform actions of the system is not developed for the collaboration system yet. In fact, the only actions that the collaborations can perform at this point are sending notifications to users. When such an API is completed, fully functional CWRC collaborations can be specified and put into operation.

# Chapter 7

# Conclusion

In this work, we studied a prevalent type of collaborative work, namely resource-centric collaborations, in which a group people collaborate in order to develop (i.e., create and update) shared resources. We identified the characteristic properties of resource-centric collaborations, with semi-structured process model and human-driven progress as the most significant among them. We reviewed the classic systems aimed at managing workflows and collaborative work, and argued that these systems are not appropriate for supporting this type of collaborations as they are either too rigid or not powerful enough. Therefore, there is a need for solutions to support resource-centric collaborations, which adequately balance coordination with flexibility. In addition, since the various activities of resource-centric collaborations are typically supported by web-based systems, these solutions should also consider the integration requirements of web-based systems.

As a solution, we introduced a service-oriented approach for supporting resource-centric collaboration. Our approach aims at coordinating people, tools, and services in service-oriented resource-centric environments. Our solution includes

- a language for specifying resource-centric collaborations,
- a workflow engine for enacting and managing collaborations at run time, and
- a systematic methdology for integrating our engine with other interactive systems used by users to manipulate resources.

Our system satisfies the requirements of resource-centric collaborations and can be integrated with any ecosystem of systems and tools in order to add the support of the collaborations to them. In the chapters of this thesis, we provided a complete specification of our collaboration language including its syntax and semantics, explained the architecture of our collaboration system, and described of our integration model. Finally, we provided two case studies as the applications of our system in collaborative projects.

The two case studies should be seen as the evidence of applicability of our approach and system in real-world projects. The CWRC project is mainly focused on managing CWRC documents; the documents are, in fact, the resources around which the collaboration are defined. The documents are manipulated using the REST interfaces provided by the CWRC repository. The documents are developed or modified by the resource-centric collaboration defined in CWRC. In fact, each type of collaborations facilitates and manages the collaboration of people on a specific type of document through its life-cycles. GRAND is a network of researchers and the GRAND Forum is responsible for managing the information about the participating researchers, their products, their projects, and the relationships among them; this information is accessible and modifiable through REST services, implemented by the GRAND Forum. The resource models of the CWRC platform and the GRAND Forum are different as are their architectures. The CWRC system is composed of a set of tools integrated to a central Fedora document repository, while the GRAND Forum is built on Mediawiki and a dedicated MySQL database management system. Although the CWRC platform and the GRAND Forum are different in many aspects, our collaboration approach and system was employed similarly in both cases using our integration model. In general, we believe that our system can be integrated with any ecosystem of tools and properly supports the required resource-centric collaborations, assuming that the ecosystem provides the components assumed by our integration model.

Our collaboration framework was developed according to the REST architectural style. Particularly, the collaboration system realizes resource-centric collaborations by mainly orchestrating REST services; then, the system publishes the resulted collaborations as REST services too. Therefore, our collaboration system can be seen as a system which enables REST service composition in a process-oriented manner. Furthermore, the REST principles were used as guidelines in the design of our resource-centric collaboration approach; specially, simplicity and flexibility were among the main design goals of our approach. The system is originally developed to support resource-centric collaborations; however, the range of its application can be wider than only this type of collaborative workflows. In fact, our approach and system would be appropriate to be support any types of workflows which conforms to the properties of REST architectural style in resource-centric environments such as the World Wide Web. For example, our work can also be used for enabling end-user service mash-ups.

In the design of our language, we deliberately did not include some features which can be typically found in web-service orchestration languages such as BPEL. For example, we ignored exception handling and transactional support. In fact, we tried to identify the essential language features and only provide native support for them. We made this decision according to the properties and requirements of resource-centric collaborations; specifically simplicity was more important than expressiveness of the collaboration language. In addition, resource-centric collaborations have simple structure and do not usually have any complex processing. In uncommon situations, where more complex processing is required, it can be addressed by delegating the processing to external services. In this way, the capabilities of the language do not suffer while the core of the language stays lightweight. we should also mention that we do not claim our language natively supports all of the common requirements. Actually, we designed the language based on sample resource-centric collaborations we explored and the conceptualization of the collaborations we developed. For example, manipulating JSON data is often required when working with REST services, so adding the native support for working with JSON data can be considered as an improvement to our language.

## 7.1 Future Work

There are some other improvements that can be applied on various aspects of the collaboration framework, specially the usability of the system. Currently, collaborations are specified in our collaboration language which has a scripting-style textual language. The users need to write the collaboration specification using our Eclipse-based collaboration editor. We greatly tried to design our language such that it can be used by less technical users. In addition, we embedded some facilitation and assistance for authoring collaboration specifications in the collaboration editor. However, it still makes a notable difference whether the person who is responsible for specifying the collaborative process is familiar with the scripting paradigm or not. Therefore, a graphical approach for writing collaborations would make the collaboration specification process much easier to learn and use. Consequently, developing a graphical notation for specifying collaborations and a graphical collaboration editor which supports the graphical notation would be an important future extension to the system. It should be considered that the graphical notation should be developed on top of the textual collaboration specification language; in other words, a valid graphical specification should be translatable to a valid textual specification (to be precise, a valid graphical specification may be translatable to many valid textual specifications, but all of them must have exactly the same executing logic). Accordingly, the graphical editor is responsible for translating the graphical specifications to textual specifications which can be then compiled by the textual collaboration specification compiler into executable specifications.

During our work on CWRC and GRAND projects, we discovered that collaborative projects usually have many similar collaborations such as creating a report or publishing an artifact. In fact, the variants of some popular collaborations can be found in almost all of the collaborative systems. In addition, many collaborations in a project or across various projects share some common collaboration fragments such as accept/reject fragment for supervisory decision making or voting fragment for distributed decision making. We believe that identification and classification of resource-centric collaboration patterns and templates (which can respectively be achieved by generalization of common fragments and collaborations) will significantly contribute to this field. As the next step, the collaboration languages and the workflow editors of collaborative systems can provide the support for these patterns, such as suggesting the patterns or facilitating their implementations.

Resource-centric collaborations typically have some steps which require asking questions from users and collecting the responses. The responses are usually used in the control elements in order to coordinates the operations of the collaborations. In fact, this is the main way by which the collaborations can interact with human participants. One of the most common scenarios involves collecting the decision of a user on one particular situation; for example, asking a supervisor to either accept or reject a project-report. These types of operations almost exist in all of the CWRC and GRAND collaborations we studied and/or developed; and in general, they are of the main constructs of resource-centric collaborations. For both CWRC and GRAND projects, we developed notification mechanisms presented as notification dashboards through which the questions are sent to the users, and the responses are collected and informed to the collaboration engine. For each project, based on the requirements of the project and its collaborations, we defined and implemented some questions. We implemented the questions such that they are not specific only to one particular collaboration but could be used in any collaborations of the project. However, it is possible to go beyond that and generalize the common questions to be reusable by different resource-centric collaboration projects. Yes-no question is probably the simplest and most common type of questions which is used in almost every resource-centric collaboration project. There are other types of questions which appear in similar forms in different projects. Therefore, it is valuable to identify these common types of questions, more generally common types of user interactions, and extends the collaboration language and system to support these questions/user interactions as first class citizens.

# Bibliography

[1] Apache activemq, November 2012. http://activemq.apache.org/.

[2] Canadian writing research collaboratory (cwrc), November 2012. http://cwrc.ca/.

[3] Fedora commons repository, June 2012. http://fedora-commons.org/.

[4] Google code blog, November 2012. http://google-code-updates.blogspot.ca/2009/03/introducing-labs-for-google-code.html.

[5] Grand nce, November 2012. http://grand-nce.ca/.

[6] Ibm websphere process server, November 2012. http://www-01.ibm.com/software/integration/wps/.

[7] Mediawiki, November 2012. http://www.mediawiki.org/wiki/MediaWiki.

[8] Oracle bpel process manager, November 2012. http://www.oracle.com/technetwork/middleware/bpel/overview/.

[9] Xtext framework, June 2012. http://www.eclipse.org/Xtext/.

[10] Yahoo pipes, November 2012. http://pipes.yahoo.com/pipes/.

[11] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. Konig, F. Leymann, R. Muller, G. Pfau, K. Ploesser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller. Web services human task (ws-humantask), version 1.0. 2007.

[12] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. Konig, F. Leymann, R. Muller, G. Pfau, K. Ploesser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller. Ws-bpel extension for people (bpel4people), version 1.0. 2007.

[13] Djamal Benslimane, Schahram Dustdar, and Amit Sheth. Services mashups: The new generation of web applications. *IEEE Internet Computing*, 12(5):13–15, September 2008.

[14] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In *Proceedings of the 5th international conference on Business process management*, BPM'07, pages 288–304, Berlin, Heidelberg, 2007. Springer-Verlag.

[15] Thomas Burkhart and Peter Loos. Flexible business processes - evaluation of current approaches. In *Proceedings of Multikonferenz Wirtschaftsinformatik 2010*, 2010.

[16] Jorge Cardoso. *Semantic Web Services: Theory, Tools and Applications*. IGI Publishing, Hershey, PA, USA, 2007.

[17] Steinar Carlsen, John Krogstie, and Odd Ivar Lindland. Evaluating flexible workflow systems. In *Proceedings of the 30th Hawaii International Conference on System Sciences: Information Systems Track-Collaboration Systems and Technology - Volume 2*, HICSS '97, pages 230–, Washington, DC, USA, 1997. IEEE Computer Society.

[18] Francisco Curbera, Matthew Duftler, Rania Khalaf, and Douglas Lovell. Bite: Workflow composition for the web. In *Proceedings of the 5th international conference on Service-Oriented Computing*, ICSOC '07, pages 94–106, Berlin, Heidelberg, 2007. Springer-Verlag.

[19] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human-Computer Interaction*. Prentice Hall., 1998.

[20] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, August 2005.

[21] Clarence Ellis, Karim Keddara, and Grzegorz Rozenberg. Dynamic change within workflow systems. In *Proceedings of conference on Organizational computing systems*, COCS '95, pages 10–21, New York, NY, USA, 1995. ACM.

[22] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[23] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[24] Christian Fritz, Richard Hull, and Jianwen Su. Automatic construction of simple artifact-based business processes. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 225–238, New York, NY, USA, 2009. ACM.

[25] J. Grudin. Computer-supported cooperative work: History and focus. *Computer*, 27(5):19–26, 1994.

[26] Petra Heinl, Stefan Horn, Stefan Jablonski, Jens Neeb, Katrin Stein, and Michael Teschke. A comprehensive approach to flexibility in workflow management systems. *SIGSOFT Softw. Eng. Notes*, 24(2):79–88, March 1999.

[27] David Hollingsworth. *The Workflow Reference Model*. Workflow Management Coalition, Hampshire, 1995.

[28] Ta'id Holmes, Martin Vasko, and Schahram Dustdar. Viebop: Extending bpel engines with bpel4people. In *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, PDP '08, pages 547–555, Washington, DC, USA, 2008. IEEE Computer Society.

[29] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems*, OTM '08, pages 1152–1163, Berlin, Heidelberg, 2008. Springer-Verlag.

[30] S. Jablonski and C. Bussler. Workflow management: modeling concepts, architecture and implementation. 1996.

[31] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. Von Riegen, P. Schmidt, and I. Trickovic. Ws-bpel extension for people (bpel4people). 2005.

[32] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42(3):428–445, July 2003.

[33] M.P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 3–12. IEEE, 2003.

[34] Cesare Pautasso. Composing restful services with jopera. In *International Conference on Software Composition 2009*, volume 5634, page 142?159, Zurich, Switzerland, July 2009. Springer.

[35] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, October 2003.

[36] Stefan Pietschmann, Vincent Tietz, Jan Reimann, Christian Liebing, Michel Pohle, and Klaus Meissner. A metamodel for context-aware component-based mashup applications. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications &#38; Services*, iiWAS '10, pages 413–420, New York, NY, USA, 2010. ACM.

[37] Florian Rosenberg, Francisco Curbera, Matthew J. Duftler, and Rania Khalaf. Composing restful services and collaborative workflows: A lightweight approach. *IEEE Internet Computing*, 12(5):24–31, 2008.

[38] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil M. P. van der Aalst. Process flexibility: A survey of contemporary approaches. In Jan L. G. Dietz, Antonia Albani, and Joseph Barjis, editors, *CIAO! / EOMAS*, volume 10 of *Lecture Notes in Business Information Processing*, pages 16–30. Springer, 2008.

[39] Nelly Schuster, Raffael Stein, Christian Zirpins, and Stefan Tai. A service mashup tool for open document collaboration. In *Proceedings of the 8th International Conference on Service Oriented Computing*, pages 713–714, 2010.

[40] Nelly Schuster, Christian Zirpins, and Ulrich Scholten. How to balance flexibility and coordination? service-oriented model and architecture for document-based collaboration on the web. In *SOCA*, pages 1–9, 2011.

[41] Nelly Schuster, Christian Zirpins, Stefan Tai, Steve Battle, and Nils Heuer. A service-oriented approach to document-centric situational collaboration processes. In *Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, WETICE '09, pages 221–226, Washington, DC, USA, 2009. IEEE Computer Society.

[42] Wei Shi, Jian Wu, Shaolin Zhou, Ling Zhang, Yuyu Yin, and Zhaohui Wu. Facilitating the flexible modeling of human-driven workflow in bpel. In *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops*, AINAW '08, pages 1615–1624, Washington, DC, USA, 2008. IEEE Computer Society.

[43] OASIS Web Services Business Process Execution Language (WSBPEL) TC. Business process execution language for web services (bpel) version 2.0. 2007.

[44] Technical Architecture Team. *ebXMLGlossary*. 2001.

[45] J. Thomas, F. Paci, E. Bertino, and P. Eugster. User tasks and access control overweb services. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 60–69. IEEE, 2007.

[46] Wil van der Aalst and Kees van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2004.

[47] Erik Wittern, Nelly Schuster, Jörn Kuhlenkamp, and Stefan Tai. Participatory service design through composed and coordinated service feature models. In *Proceedings of the 10th International Conference on Service Oriented Computing*, pages 158–172, 2012.

# Appendix A

# Translation to Java

The specifications written in our collaboration language are not understandable by the engine as the way they are. Our collaboration language is designed to be easy to program for the users, whereas the engine only understands some specific structured Java programs that include logic about the collaborations and their interactions. Therefore, the first obvious step in providing the engine with the understandable version of collaboration and configuration specifications is to transform them into their Java representations. In this chapter, we present how the specifications written in our collaboration language can be translated to target structured Java programs and packages.

We organized our translation procedure into two sections, translation of a configuration specification, and translation of collaboration specification. For each specification, we explore the elements defined in each of the specifications and explain how these declarations and logic's can be translated to a Java program that is understandable by the collaboration engine. For each element, we provide the list of rules that are followed by our translator and provide an example of its translation from its specification in our collaboration language to the target Java program. Since the Type element has been used in translation of both specifications, we presented he details about translation Type before everything else.

## A.1 Types

Each Type defined in our collaboration language should be transformed to a data type understandable by our collaboration engine. These data types have been implemented as individual java classes in the engine package, and the translator only employs them in the target Java collaboration program. For example, wherever there is a field or variable of type User in our collaboration or configuration specification in our collaboration language, the translator transforms it to UserType, which is defined for the engine. The data types defined for Boolean, Integer, Time, String, Strings, User and Users in our collaboration language are respectively BooleanType, IntegerType, TimeType, StringType, StringsType, UserType and UsersType in collaboration engine package.

## A.2 Configuration Specification

While transforming the configuration specification a new Java project needs to be created, including classes for configuration elements. The project, named ConfigurationProject, includes a Java class for each event declared in the original specification, a single Java class containing the translated Roles as its methods, a single Java class containing the translated Relations as its methods, a single Java class containing the translated Services as its methods. Each of the elements defined in a configuration specification, including Event, Role, Relation and Service, needs to be transformed to classes or methods in the target format. In the remainder of this section we explore each of these elements and explain how each of them should be translated.

### A.2.1 Event Declaration

For each Event declared in the Configuration Specification, an individual Java class should be built in the ConfigurationProject. All event classes are subclasses of AbstractEvent.

The created class will be named according to the Event Name, e.g. if the Event Name is “Update” the class would be named as UpdateEvent. The parameters of the events in the Event Declaration will be transformed as the fields of the class. The class has a constructor receiving a parameter referring to the sender of the event, and a parameter for each of the fields with the same name and translated Type. Moreover, for each parameter in the Event Declaration, i.e. each field of the event class, a getter method should be defined in the class. Beside these, all event classes should include a method called “visit” that is needed for handling the event and implementing the visitor pattern. The signature and the body of this method is the same for all events.

For example, the Event Declaration

```
Event Update (String name*, Integer number);
```

will be translated to the event class that is presented in the following.

```
public class UpdateEvent extends AbstractEvent {
  private static final String eventName = "Update";
  private StringType name;
  private IntegerType number;

  public EventUpdate (UserType sender, StringType name, IntegerType number) throws
       WfmsException{
    super(eventName, sender);
    if(name==null){
      throw new WfmsException("...ExceptionMessage...");
    }
    this.name= name;
    this.number= number;
  }

  public StringType getName() {
    return name;
  }

  public IntegerType getNumber() {
    return number;
  }
  @Override
  public State visit(EventHandler eventHandler) throws WfmsException{
    return ((CollaborationEventHandler)eventHandler).handleEvent(this);
  }
}
```

### A.2.2 Service Declaration

For each Service declared in the Configuration Specification, a method should be created and added to the Services class in the ConfigurationProject. ConfigurationProject includes a class, named Services, containing all methods for the defined services in the Configuration Specification.

The created method should be a public static method that is named according to the Service Name, e.g. if the Service Name is “AddMember” the method would be named as AddMemberService. The return type of the method should be the translated form of the return Type of the Service. The parameters of the method should be the translated form of all the parameters presented in the Service Declaration, with the same names and transformed Types, and also in the same order. At the beginning of the method, a Set of Parameters should be created and for each parameter defined for the method, an element should be added to the Set, using two values for the element: (1) the a string literal with value of the name of the parameter, and (2) the result of calling toString() method on that parameter. After adding all parameters to the Parameter Set, the RestClient should call the service using either POST or GET using the a string literal including the exact URL presented in the Service Declaration, and the created Set of Parameters. If the Service Declaration uses POST then the RestClient also uses POST, and it uses GET if it is specified in the corresponding Service Declaration. Then if the response of the client is as desired, the method returns a new instance of the translated return Type of the service with using the entity included in the response. If the status of the response is not as desired, the method should throw an exception.

For example, the Service Declaration

```
String POST AddMember(User user, String project) : "...url...";
```

will be translated to the method that is presented in the following.

```
public static TypeString AddMemberService (UserType user, StringType project)
    throws WfmsException{
  Set<Parameter> params= new HashSet<Parameter>();
  params.add(new Parameter("user", user.toString()));
  params.add(new Parameter("project ", project.toString()));
  Response response = RestClient.POST("...url...", params);
  if(response.getStatus() < 200 || response.getStatus()>299)
    throw new WfmsException("...ExceptionMessage...");
  return new StringType (response.getEntity());
}
```

### A.2.3 Role Declaration

For each Role declared in the Configuration Specification, two methods should be created and added to the Roles class in the ConfigurationProject. ConfigurationProject includes a class, named Roles, containing all methods for the defined roles in the Configuration Specification.

The first method should be a public static method that is named as “Is” followed by the Role Name followed by “Role”, e.g. if the Role Name is “Manager” the method would be named as IsManagerRole. The return type of the method should be BooleanType since the method returns True if the inputted user has the role and returns False otherwise. The method has an input parameter of type UserType named the same as the Parameter Name. At the beginning of the method, a Set of Parameters should be created and for the only input parameter of the method added to the Set, using two values for the element: (1) the a string literal with value of the name of the parameter, and (2) the result of calling toString() method on that parameter. Then, the RestClient should call the service using POST and a string literal including the first URL presented in the Role Declaration, and the created Set of Parameters. If the response of the client is as desired, the method returns a new BooleanType using the entity included in the response. If the status of the response is not as desired, the method should throw an exception.

The second method should be a public static method that is named as “All” followed by the Role Name followed by “Role”, e.g. if the Role Name is “Manager” the method would be named as AllManagerRole. The return type of the method should be UsersType since the method returns a collection including all users having that Role. The method does not have any input parameters. At the beginning of the method, a Set of Parameters should be created but no elements should be added to that since the method does not have any inputs. Then, the RestClient should call the service using GET and a string literal including the second URL presented in the Role Declaration, and the created Set of Parameters. If the response of the client is as desired, the method returns a new UsersType using the entity included in the response. If the status of the response is not as desired, the method should throw an exception.

For example the Role Declaration:

```
Role Manager (user) : "...url1...", "...url2...";
```

will be translated to the the methods that are presented in the following.

```
public static BooleanType IsManagerRole (UserType user) throws WfmsException{
  Set<Parameter> params= new HashSet<Parameter>();
  params.add(new Parameter("user", user.toString()));
  Response response = RestClient.GET("...url1...", params);
  if(response.getStatus() < 200 || response.getStatus()>299)
    throw new WfmsException("...ExceptionMessage...");
  return new BooleanType (response.getEntity());
}

public static TypeUsers AllManagerRole () throws WfmsException{
  Set<Parameter> params= new HashSet<Parameter>();
  Response response = RestClient.GET("...url2...", params);
  if(response.getStatus() < 200 || response.getStatus()>299)
    throw new WfmsException("...ExceptionMessage...");
  return new UsersType (response.getEntity());
}
```

### A.2.4 Relation Declaration

For each Relation declared in the Configuration Specification, three methods should be created and added to the Relations class in the ConfigurationProject. ConfigurationProject includes a class, named Relations, containing all methods for the defined relations in the Configuration Specification.

The first method should be a public static method that is named as “Is” followed by the Relation Name followed by “Relation”, e.g. if the Relation Name is “Leads” the method would be named as IsLeadsRelation. The return type of the method should be BooleanType since the method returns True if the relation is held between the inputted parameters and returns False otherwise. The method has two input parameters which are in the translated forms of the presented parameters in the Relation Declaration. Each parameter should be named the same as its corresponding Parameter Name, and its type is the translated form of the presented Type for the parameter. At the beginning of the method, a Set of Parameters should be created and for the only input parameter of the method added to the Set, using two values for the element: (1) the a string literal with value of the name of the parameter, and (2) the result of calling toString() method on that parameter. Then, the RestClient should call the service using GET and a string literal including the first URL presented in the Relation Declaration, and the created Set of Parameters. If the response of the client is as desired, the method returns a new BooleanType using the entity included in the response. If the status of the response is not as desired, the method should throw an exception.

The second method should be a public static method that is named as “FindX” followed by the Relation Name followed by “Relation”, e.g. if the Role Name is “Leads” the method would be named as FindXLeadsRelation. The return type of the method should be the translated collection type of the first parameter defined in the Relation Declaration, since the method returns a collection including all left-hand side elements, such that the relation is held between those elements and the inputted parameter. For example, if the type of the first parameter is String, the return type of the method would be StringsType. The method has one input parameter that is the translated forms of second parameter presented in the Relation Declaration. At the beginning of the method, a Set of Parameters should be created and the only parameter of the method should be added to the Set. Then, the RestClient should call the service using GET and a string literal including the second URL presented in the Relation Declaration, and the created Set of Parameters. If the response of the client is as desired, the method returns a new object of translated collection type of the second parameter, using the entity included in the response. If the status of the response is not as desired, the method should throw an exception.

The third method should be a public static method that is named as “Find” followed by the Relation Name followed by “XRelation”, e.g. if the Role Name is “Leads” the method would be named as FindLeadsXRelation. The return type of the method should be the translated collection type of the second parameter defined in the Relation Declaration, since the method returns a collection including all right-hand side elements, such that the relation is held between the inputted parameter and those elements. For example, if the type of the second parameter is String, the return type of the method would be StringsType. The method has one input parameter that is the translated forms of first parameter presented in the Relation Declaration. At the beginning of the method, a Set of Parameters should be created and the only parameter of the method should be added to the Set. Then, the RestClient should call the service using GET and a string literal including the second URL presented in the Relation Declaration, and the created Set of Parameters. If the response of the client is as desired, the method returns a new object of translated collection type of the second parameter, using the entity included in the response. If the status of the response is not as desired, the method should throw an exception.

For example Relation Declaration:

```
Relation Leads(User user, String pid) : "...url1...", "...url2...";
```

will be translated to the methods that are presented in the following.

```
public static BooleanType IsLeadsRelation (UserType user, StringType pid) throws
    WfmsException{
  Set<Parameter> params= new HashSet<Parameter>();
  params.add(new Parameter("user", user.toString()));
  params.add(new Parameter("pid", pid.toString()));
  Response response = RestClient.GET("...url1...", params);
  if(response.getStatus() < 200 || response.getStatus()>299)
    throw new WfmsException("...ExceptionMessage...");
  return new BooleanType (response.getEntity());
}

public static UserType FindXLeadsRelation (StringType pid) throws WfmsException{
  Set<Parameter> params= new HashSet<Parameter>();
  params.add(new Parameter("pid", pid.toString()));
  Response response = RestClient.GET("...url2...", params);
  if(response.getStatus() < 200 || response.getStatus()>299)
    throw new WfmsException("...ExceptionMessage...");
  return new UserType (response.getEntity());
}

public static StringType FindLeadsXRelation (UserType user) throws WfmsException{
  Set<Parameter> params= new HashSet<Parameter>();
  params.add(new Parameter("user", user.toString()));
  Response response = RestClient.GET("...url2...", params);
  if(response.getStatus() < 200 || response.getStatus()>299)
    throw new WfmsException("...ExceptionMessage...");
  return new StringType (response.getEntity());
}
```

## A.3 Collaboration Specification

While transforming a collaboration specification, a new package is created, including classes for collaboration elements and logic. The package should be named after the Collaboration Name, e.g. if the Collaboration Name is “Reporting” the name of the corresponding project would be ReportingCollaborationPackage. The package includes Java classes including the logic and elements of the collaboration, each of them translated into a specific format. These classes consist of (1) the “Data” class including all data elements of the collaboration, such as its fields and sub-collaborations; (2)

one class for each state defined in a state-based collaboration specification; (3) the “Collaboration” class including the main executable method of the collaboration and all construction procedures (i.e. constructor methods) that are obtained from Entry Declarations.

In the remainder of this section we explore each of the collaboration specification elements and explain how they should be transformed into their Java form and be organized in the mentioned classes.

### A.3.1 Sub-Collaboration Declaration & Field Declaration

All Sub-Collaborations and Fields declared in the Collaboration Specification, will be translated as fields of CollaborationData class in the corresponding collaboration project. Therefore, the CollaborationData class should be created as a subclass of AbstractData, and should include the translated forms of all collaboration fields and sub-collaborations and their corresponding getter and setter methods. Moreover, they should be two methods, named serialize and deserialize, that will be used for storing and retrieving collaboration instances to/from Instances Database. The body of these methods should also be implemented based on the declared fields and sub-collaborations in the Collaboration Specification.

For each Field Declaration in the collaboration, there should be a private field created for CollaborationData, with the exact same name but the translated form of its original Type. For each Sub-Collaboration Declaration in the collaboration, there should be a private field created for CollaborationData, with the exact same name and type String. For each of these declared fields in CollaborationData the appropriate getter and setter methods should be added to the CollaborationData class. The method serialize is a public method with the return type of String. In this method, we first define an ArrayList of String, and then for each Field and Sub-Collaboration declared we add its String representation to the list. Then the method should return the result of serializing the list. The method deserialize is a public void method which received an input parameter of type String. In this method, we first evaluate the deserialized ArrayList of all parameters, and then for each Field and Sub-Collaboration declared we update the value of the corresponding field in CollaborationData according to the elements of the deserialized arrayList.

For example the Sub-Collaboration and Field Declarations

```
String name;
Integer number;
DocumentCheckCollaboration wf;
```

will be translated in a data class that is presented in the following.

```
public class CollaborationData extends AbstractData {
  private StringType name;
  private IntegerType number;
  Private String wf;


  public StringType getName() {
    return name;
  }
  public void setName(StringType name) {
    this.name = name;
  }
  public IntegerType getNumber() {
    return number;
  }
  public void setNumber(IntegerType number) {
    this.number = number;
  }
  public String getWf() {
    return wf;
  }
  public void setWf(String wf) {
    this.wf = wf;
  }

  @Override
  public String serialize() {
    ArrayList<String> params= new ArrayList<String>();
    params.add(name.ToString());
    params.add(number.ToString());
    params.add(wf);
    return Serializer.serialize(params);
  }

  @Override
  public void deserialize(String data) {
    ArrayList<String> params = Serializer.deserialize(data);
    name = new StringType(params .get(0));
    number = new IntegerType(params .get(1));
    wf = params .get(2);
  }
}
```

### A.3.2 State Declaration

For each state declared in the Collaboration Specification, an individual Java class should be built in the corresponding collaboration project. All state classes are subclasses of AbstractState class. The created class will be named according to the State Name, e.g. if the State Name is “Draft” the class would be named as DraftState. The state class has two fields storing the name of the state and also a Boolean showing if the state is a Final State or not. The body of each State Declaration may include a list of Event-Handlers and Time-Handlers. Therefore, the corresponding state class has an inner class containing all translated forms of Event-Handles and Time-Handlers defined for that state, i.e. one method for each handler is included in the inner class.

For example the State Declaration:

```
State Draft {
...event-handlers & time-handlers...
}
```

will be translated to the state class that is presented in the following.

```
public class DraftState extends AbstractState {

  public static final String stateName = "Draft";
  public static final boolean isFinal = false;

  class EventHandler extends CollaborationEventHandler {
```

```
  ...translated event-handlers & time-handlers ...
 }

 public DraftState() {
   super(name, isFinal);
   setEventHandler(new EventHandler());
 }

 public DraftState(Data data) {
   this();
   setData(data);
 }

}
```

### A.3.3 Entry Declaration

For each Entry Declaration declared in the body of a State, a constructor should be created in Collaboration class of the CollaborationPackage. The input parameter of the created constructor is the translated Name of the corresponding Event, which is defined in the ConfigurationProject.

At the beginning of the created constructor, for each of the Roles listed in the Entry Declaration we check if the sender of the event has that Role or not, i.e. by calling the translated Is method of each of roles on the event sender, e.g. Roles.IsStudnetRole(e.Sender). If the event sender does not have any of the listed Roles, the method throws an exception; otherwise it performs the translated Block of code presented in the body of the Entry Declaration. Note that the State Change presented in the Block of Entry Declaration should be translated to a method call to set the state of the collaboration.

For example, the Entry Declaration

```
Entry Initiate [Student, Manager] {
  ...Body...
}
```

will be translated to

```
public AbstractState entry(InitiateEvent  event) throws WfmsException {
  if ( !Roles.IsStudnetRole(e.Sender).toBoolean() || !Roles.IsManagerRole(e.Sender
      ).toBoolean())
    throw new WfmwException("...ExceptionMessage...");
  ...TranslatedBody...
}
```

### A.3.4 Event Handler

For each Event Handler declared in the body of a State, a method should be created in the Event Handler inner-class of the corresponding state class. The created method should be an event handler of which return type is AbstractState. The method will be named as handleEvent and the type of its input parameter is the translated Name of the corresponding Event, which is defined in the ConfigurationProject.

At the beginning of the method, for each of the Roles listed in the Event Handler we check if the sender of the event has that Role or not, i.e. by calling the translated Is method of each of roles on the event sender, e.g. Roles.IsStudnetRole(e.Sender). If the event sender does not have any of the listed Roles, the method throws an exception; otherwise it performs the translated Block of code presented in the body of the Event Handler.

For example, the Event Handler

```
@Submit [Student, Manager] {
  ...Body...
}
```

will be translated to

```
public AbstractState handleEvent(SubmitEvent  event) throws WfmsException {
  if ( !Roles.IsStudnetRole(e.Sender).toBoolean() || !Roles.IsManagerRole(e.Sender
      ).toBoolean())
    throw new WfmwException("...ExceptionMessage...");
  ...TranslatedBody...
}
```

### A.3.5 Time Handler

For each Time Handler declared in the body of a State, a method should be created in the Time Handler inner-class of the corresponding state class. The created method should be an event handler of which return type is AbstractState. The method will be named as handleEvent and the type of its input parameter is TimeEvent which is defined in the collaboration engine. The method should contain the translated Block of code presented in the body of the Time Handler.

Moreover, for each Time Handler declared in the body of a State, a line of code should be added to the constructor of the corresponding state.

For example, the Time Handler

```
On timeField {
  ...Body...
}
```

The method to be added to Time Handler inner-class of the corresponding state is

```
public AbstractState handleEvent(TimeEvent event) throws WfmsException {
  ...TranslatedBody...
}
```

and the line of code included in the constructor of the corresponding state will be

```
TimeEventManager.getInstance().addEvent(getData().getWfId, getData().getTimeField
    ());
```

### A.3.6 Block

Any Block presented in the collaboration specification should be transformed to a block of java statements. All lines of code in the Block should be translated to the targeted language and this translation should be performed in order of the occurrences the commands in the Block. Each line should be translated based on what type of command it is, either an assignment statement, If Statement, Variable Declaration and etc.; therefore, the appropriate translation steps should be taken.

### Variable Declaration

A variable declared in a collaboration specification should be translated into a variable in the target Java program. The name of the translated variable will be the same as the Variable Name, but its type should be the translated form of its original Type.

For example, the Variable Declaration

```
Users usersVar;
```

will be translated to

```
UsersType usersVar;
```

### Assignment Statement

Any Assignment Statement presented in the collaboration specification should be transformed to a Java assignment statement. The translated form of Assignment Statement should be the translated form of the left-hand side Expression followed by character “=”, followed by the translated form the right-hand side Expression. Variable Assignment Statement

```
user1Var = user2Var
```

will be translated to

```
user1Var=user2Var
```

### If Statement

Any If Statement presented in the collaboration specification should be transformed to a java if-statement. The condition of the translated if-statement should be the translated form of the original Expression provided as the Condition of the If Statement. Therefore, the translated form of If Statement should be “if (” followed by the translated form of its conditional Expression followed by “)”, followed by the translated form of its Block of code. If the If Statement includes an Else part, the translated if-statement should be followed by “else” followed by the translated form of its following Block.

For example, the If Statement

```
If (booleanVar) {
  ...Body1...
}
Else {
  ...Body2...
}
```

will be translated to

```
if (booleanVar) {
  ...TrasnlatedBody1...
}
else {
  ...TrasnlatedBody2...
}
```

### While Statement

Any While Statement presented in the collaboration specification should be transformed to a java while-statement. The condition of the translated while-statement should be the translated form of the original Expression provided as the Condition of the While Statement. Therefore, the translated form of While Statement should be "while (" followed by the translated form of its conditional Expression followed by ")", followed by the translated form of its Block of code. For example, the While Statement

```
While(booleanVar) {
  ...Body...
}
```

will be translated to

```
while(booleanVar) {
  ...TranslatedBody...
}
```

### Foreach Statement

Any Foreach Statement presented in the collaboration specification should be transformed to a java for-statement. The translation of Foreach is performed according to the type of the targeted collection to iterate. Since the allowed types of collections in our collaboration language are Users and Strings, the type of the Variable Name defined in Foreach Statement should be either User or String; it should be User if the type of targeted collection is Users, and String if the type of targeted collection is Strings. Accordingly, our translated for-statement should iterate over a list of UserType or StringType elements, which is provided as an Expression in the original program. Therefore, the translated form of Foreach Statement should be "for (" followed by the translated form of the type of the elements (either StringType or UserType) in the targeted element and the exact Variable Name presented in the Foreach Statement, the character ":", the translated form of the presented Expression as the targeted collection followed by ".getList()"and the character ")", all followed by the translated form of its Block of code. For example, the Foreach Statement

```
Foreach(userVar in usersVar) {
  ...Body...
}
```

will be translated to

```
for(UserType userVar : usersVar.getList()) {
  ...TranslatedBody...
}
```

### Service Invocation

For any Service Invocation occurred in the collaboration specification the translated method of the corresponding should be called in the Java program. Since all translated service methods are resided in the Services class of ConfigurationProject, the method call would be “Services.” Followed by the name of the translated corresponding method. The parameters needed for this method call would be the translated form of the Expression List presented in Service Invocation.

For example, the Service Invocation

```
AddMember(userVar, nameVar)
```

will be translated to

```
Services.AddMemberService (userVar, nameVar)
```

Note that Any Expression List presented in the collaboration specification should be transformed to a list of translated expressions. The elements of the transformed expressions list should be separated by the character “,” and should be presented in the same order as the original expressions have occurred in the specification.

### Exception

Any Exception presented in the collaboration specification should be transformed to a Java throw exception statement, throwing an instance of WfmsException with the intended error-message. Therefore, The translated form of the Exception would be “throw new WfmsException( ” followed by the translated form of the Expression presented as the error-message, followed by character “)”.

For example, the Exception

```
Exception(messageVar);
```

will be translated to

```
throw new WfmsException(messageVar);
```

### State Change

State Change should be translated to its target Java statement based on where it has occurred. Any State Change presented in a Event-Handler (or Time-Handler) should be transformed to a Java statement that returns an instance of a state class created for a state-based collaboration. The type of the state object to be instantiated and returned should be the translated form of the State Name presented in State Change. On the other hand, any State Change presented in an Entry Declaration should be transformed to a Java method call to set the state of a state-based collaboration, i.e. setState method. The name of the state to be set as the current State of the collaboration should be obtained State Name presented in State Change.

For example, the State Change

```
To(Draft);
```

in a Event-Handler (or Time-Handler) and in a will be translated to

```
return new DraftState(getData());      // if the State Change belongs to a Event-
    Handler (or Time-Handler)
SetState(getData());                 // if the State Change belongs to an Entry
    Declaration
```

#### Terminate

Any Terminate command presented in the collaboration specification should be transformed to a Java statement that return an instance of the TerminateState, which is a state class created for a rule-based collaboration. Therefore, the command

```
Terminate;
```

presented in the collaboration specification will be translated to

```
return new TerminateState(getData());
```

#### EventTrigger

Any Event Trigger presented in the collaboration specification should be transformed to a Java method call for triggering the event. If the event should be triggered for one of the sub-collaborations the translation would be “Engine.getInstance().trigger(getData().”, followed by the getter method of the corresponding sub-collaboration, character “,” followed by an instance of the targeted Event and the translated forms of its parameters, and character “)”. For example, the Event Trigger

```
wf.Trigger(Start(nameVar, numberVar))
```

will be translated to

```
Engine.getInstance().trigger(getData().getWf(), new StartEvent(nameVar, numberVar)
    );
```

If the event should be triggered for the parent collaboration, we employ the getParent() method instead of the getter method. For example, the Event Trigger

```
Trigger(Start(nameVar, numberVar))
```

will be translated to

```
Engine.getInstance().trigger(getData().getParent(), new StartEvent(nameVar,
    numberVar));
```

### A.3.7 Expression

Any Expression presented in the collaboration specification should be transformed to a block of java statements. Depending on the type of the expression and the operations used in it, different translating steps should be applied. Note that translation should be performed in an order that the precedence of the operations is hold. Accordingly, for any compound expression, i.e. an expression including more than one operator, the expressions with the higher priority is translated first, and then its results should be used as part of the other expressions. For example, in translation of a*(b+c), first (b+c) should be translated and then its result should be used as the right-hand side expression of the multiply expression. Consequently, we follow a bottom-up approach in translating the compound expressions

**Role&Relation Expression**

For any Role&Relation Expression occurred in the collaboration specification the translated method of the corresponding Role or Relation should be called in the Java program. Any Role&Relation Expression can be one of the five following types for which different translation steps should be applied.

- If the Expression is checking if a User has a specific Role, the translation would be calling the Is method of the Role on that specific User. Accordingly, the translated form would be "Roles." followed by the translated Is method name of the Role, and the name of the translated form of the User value should be passed as the parameter to the method.

  For example, the Expression

  ```
  userVar Is Manager
  ```

  will be translated to

  ```
  Roles.IsManagerRole(userVar)
  ```

- If the Expression is looking for all Users that have a specific Role, the translation would be calling the All method of the Role. Accordingly, the translated form would be "Roles." followed by the translated All method name of the Role.

  For example, the Expression

  ```
  All Manager
  ```

  will be translated to

  ```
  Roles.AllManagerRole()
  ```

- If the Expression is checking if a Relation is held between two values, the translation would be calling the Is method of the Relation. Accordingly, the translated form would be "Relations." followed by the translated Is method name of the Relation, and the translated form of the two parameters should be passed to the method.

  For example, the Expression

  ```
  userVar Leads projectVar
  ```

  will be translated to

```
Relations.IsLeadsRelation(userVar, projectVar)
```

- If the Expression is looking for all left-hand side elements of a Relation with a right-hand side value, the translation would be calling the first Find method of the Relation. Accordingly, the translated form would be “Relations.” followed by the translated first Find method name of the Relation, and the translated form of the input parameter should be passed to the method.

  For example, the Expression

  ```
  Find (? Leads projectVar)
  ```

  will be translated to

  ```
  Relatios.FindXLeadsRelation(projectVar)
  ```

- If the Expression is looking for all right-hand side elements of a Relation with a left-hand side value, the translation would be calling the second Find method of the Relation. Accordingly, the translated form would be “Relations.” followed by the translated second Find method name of the Relation, and the translated form of the input parameter should be passed to the method.

  For example, the Expression

  ```
  Find (userVar Leads ?)
  ```

  will be translated to

  ```
  Relatios.FindLeadsXRelation(UserVar)
  ```

**Arithmetic Expression**

Each Arithmetic Expression in our collaboration language should be translated into a method call. Since Arithmetic Expressions are designed to manipulate Integers, the callee methods belong to class IntegerType, which is defined in the collaboration engine. IntegerType includes methods add, subtract, multiply and divide to implement the arithmetic operators +, -, *, / respectively. Each of these methods receives two input arguments as the operands of the arithmetic operation. Therefore, for translating Arithmetic Expressions we write “IntegerType.” followed by the name of the translated method for the presented operator, and then we provide it with the translated form of the operands (i.e. the two expressions) as its input arguments.

For example, the Arithmetic Expressions

```
Number1Var + Number2Var
Number1Var - Number2Var
Number1Var * Number2Var
Number1Var / Number2Var
```

will be respectively translated to

```
IntegerType.add(Number1Var , Number2Var)
IntegerType.subtract(Number1Var , Number2Var)
IntegerType.multiply(Number1Var , Number2Var)
IntegerType.devide(Number1Var , Number2Var)
```

**Collection Expression**

Each Collection Expression in our collaboration language should be translated into a method call. Since Collection Expressions are designed to manipulate Strings and Users, the callee methods belong to class StringsType or UsersType, which are defined in the collaboration engine, depending what the types of the operands are. StringsType and UsersType include methods addAll and removeAll to implement the collection operators + and - respectively. Each of these methods receives two input arguments as the operands of the arithmetic operation. Therefore, for translating Collection Expressions manipulating Strings we write "StringsType." followed by the name of the translated method for the presented operator, and then we provide it with the translated form of the operands (i.e. the two expressions) as its input arguments. For Collection Expressions manipulating Users the translation would be the same except that the callee method belongs to UsersType.

For example, the Collection Expressions

```
users1Var + users1Var
users1Var - users1Var
```

will be respectively translated to

```
UsersType.addAll(users1Var , users2Var)
UsersType.removeAll(users1Var , users2Var)
```

**Logical Expression**

Each Logical Expression in our collaboration language should be translated into a method call. Since Logical Expressions are designed to manipulate Strings, the callee methods belong to class BooleanType, which is defined in the collaboration engine. BooleanType includes methods and, or, and not, to implement the logical operators And, Or, ! respectively. The first two methods receive two input arguments as the operands of the logical operation, whereas the third one only needs one operand. Therefore, for translating Logical Expressions we write "BooleanType." followed by the name of the translated method for the presented operator, and then we provide it with the translated form of the operand(s) (i.e. the expression(s)) as its input argument(s).

For example, the Logical Expressions

```
Boolean1Var And Boolean2Var
Boolean1Var Or Boolean2Var
! Boolean1Var
```

will be respectively translated to

```
BooleanType.and(Boolean1Var, Boolean2Var)
BooleanType.or(Boolean1Var, Boolean2Var)
BooleanType.not(Boolean1Var)
```

### String Expression

Each String Expression in our collaboration language should be translated into a method call. Since String Expressions are designed to manipulate Integers, the callee methods belong to class String-Type, defined in the collaboration engine. StringType includes method concat to implement the string operator +. This method receives two input arguments as the operands of the string operation. Therefore, for translating String Expressions we write “StringType.” followed by the name of the translated method, i.e. “concat”, and then we provide it with the translated form of the operands (i.e. the two expressions) as its input argument(s).

For example, the String Expression

```
String1Var + String2Var
```

will be respectively translated to

```
BooleanType. concat (String1Var, String2Var)
```

### Reference Expression

Each Reference Expression in our collaboration language should be translated into a method call or a variable reference. If the Reference Expression is a Field Name in the collaboration specification, it should be translated to “getData().” Followed by the getter method defined for the corresponding field. If the Reference Expression is a Variable Name in the collaboration specification, it can be used the way it is as a reference to the corresponding Variable. If the Reference Expression is a reference to a parameter of the event, it should be translated to “event.” Followed by the getter method defined for the corresponding parameter of the event.

For example, the Reference Expressions

```
NumberField
UserVar
e.Sender
```

will be respectively translated to

```
getData().getNumberField()
UserVar
event.getSender()
```

### Literal

Any Literal presented in the collaboration specification should be translated to a Java instance of the translated Type of the Literals. A Literal of type Integer, String, or Boolean should be translated to an instance of IntegerType, StringType, or BooleanType with the same value as the Literal. In other words, an object of type IntegerType, StringType, or BooleanType should be “new”ed while passing the value of the Literal to the constructor of the corresponding class. The Literal “null” can be used in the Java program the way it is and instantiating any object, meaning that the translation of the “null” Literal is “null”.

For example, the Literals

```
1
"Hello"
True
null
```

will be respectively translated to

```
new IntegerType(1)
new StringType("Hello")
new BooleanType(true)
null
```