Abstract—In this paper we investigate whether CollabRDL conforms to the 3C model designed for groupware applications involving small groups. CollabRDL extends RDL, which is an acronym for Reuse Description Language. It is textual and executable and allows us to describe activities of the reuse process explicitly following an order. By reviewing the literature on the 3C Model and on Role Analysis, we aim to achieve our goals and have an indication of where we need to improve.

Keywords—collaboration; software reuse; role; language; CollabRDL, collaborative software engineering; 3C Model.

I. INTRODUCTION

It is difficult to describe and monitor collaboration. Conceptual models with the goal of characterizing collaboration are found in the literature [1,2,3] and serve as a basis for building and monitoring tools to support collaboration.

Modern software systems are developed by people working together since the complexity of these systems requires knowledge in various disciplines. In this context collaboration between people emerges as an important factor for the success of a software project development and therefore tools to support collaborative work are needed [4].

Furthermore, software reuse allows us to reuse the knowledge acquired in previous projects during a current project, allowing gains in quality and economy in the resources involved [5]. In this scenario, the Collaborative Software Reuse [6,7] unites the concepts of collaborative work and software development from reusable artifacts, so that the development process can be harmonious [8].

This paper briefly presents CollabRDL, a language to support collaborative reuse processes. CollabRDL extends RDL, which is an acronym for Reuse Description Language [9]. It is textual and executable and allows us to describe activities of the reuse process explicitly following an order. The main goal of this research article is to verify whether or not CollabRDL conforms to the 3C model designed for groupware applications involving small groups [1,2]. This way we will have an indication of where we need to improve it. To achieve our goals, it is necessary to review the literature on role analysis.

This paper is organized as follows. Section 2 presents CollabRDL as a language that aims to improve the software reuse process. Section 3 makes a brief literature review about roles. Section 4 checks if CollabRDL conforms to the 3C model. Section 5 shows how we can improve the collaboration support in CollabRDL. And section 6 presents our conclusions.

II. COLLABRDL AND SOFTWARE REUSE

CollabRDL is an extension of RDL to supporting collaborative work. RDL is an imperative language to describe software reuse activities. Currently there is a version to instantiate object-oriented frameworks step-by-step [9]. Generally speaking, CollabRDL is RDL plus ROLE, PARALLEL and DOPARALLEL commands. Figure 1 shows an overview of CollabRDL, where the Framework Model is usually organized in Model, View, and Controller, and each model has its requirements, therefore requiring people with particular abilities - the Experts. So activities (that instantiate the framework) are organized in Model, View, and Controller, and the Reusers are organized in Reuse Groups. After that, activities may be delegated to Reuse Groups according to their abilities and responsibilities. These activities may be performed in serial or parallel - it depends on the relationship of interdependence between activities [10].

Fig. 1. Overview of CollabRDL, inspired in Figure 2 of [27]
The most often commands used in RDL are CLASS_EXTENSION and METHOD_EXTENSION (Table 1). CLASS_EXTENSION command has three parameters: the first is to indicate the complete name of the super class; the second indicates the package name of the new class, and the last asks the developer for the name of the new class. At runtime this command creates a new class with the name answered by reuser that inherits from package.superClass.nameSuperClass in package.newClass and returns the newly created class. METHOD_EXTENSION command receives three parameters: the complete name of the super class, a class variable that represents a Class, and the method name which exists in the super class. At runtime this command creates a method in class named methodName and returns the newly created method. In [9] we can find a complete reference of RDL commands.

**TABLE I. CLASS_EXTENSION AND METHOD_EXTENSION COMMANDS**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS_EXTENSION</td>
<td>creates a new class with the name answered by reuser</td>
</tr>
<tr>
<td>METHOD_EXTENSION</td>
<td>creates a method in class named methodName</td>
</tr>
</tbody>
</table>

Now we detail the commands CollabRDL. The ROLE command was based upon the definition of roles [11] and in the workflow pattern WRP2 [12], role-based distribution. It allows delegate a block of activities to a group of people, see Table 2, It has two parameters. The analyst parameter identifies the group to which they will be offered the activities contained in the block delimited by braces, i.e., activityA and activityB. The second parameter is information that may help in performing activities. Thus, at runtime, the code of Table 2 offers to analyst group activities activityA and the activityB, and the activityB only will be offered after the completion of the activity activityA.

**TABLE II. THE ROLE COMMAND**

```
ROLE (analyst, "a comment"){
    activityA;
    activityB;
}
```

There are activities that can be performed in the same time interval by different people in Group Work. The command PARALLEL is capable of expressing blocks of activities which can be executed in the same time interval by groups of people. Its implementation meets the standards WCP2, WCP3, WCP4, WCP5 and WCP6 [12]. Table 3 presents the PARALLEL command, it indicates that at runtime will create two blocks of activities that can be executed in parallel, and they will be delegates respectively for analyst and designer groups. So activityA activity can be performed by a reuser of group analyst at the same time that a reuser of designer group can perform the activityB activity. And activityC activity could be performed only after execution of these two activities.

**TABLE III. THE PARALLEL COMMAND**

```
PARALLEL{
    FLOW (analyst, "a comment"){
    activityA;
    }
    FLOW (designer, "another comment"){
    activityB;
    }
    activityC;
}
```

In some situations, when describing a process of reuse, there is a need to repeat a sequence of activities for a number of times defined at runtime. One way to express this in RDL is through the LOOP command [9]. Table 4 illustrates this situation at runtime is emitted question "Run the block of the loop?" If the answer is yes, the activity activityA will be held for execution, and only after its execution, the question of the LOOP will be reissused. This behavior indicates that the LOOP command execution produces n code blocks serially with the determination of n at runtime, this behavior is described in the standard variant thereof serial WCP15 [12].

**TABLE IV. THE LOOP COMMAND**

```
LOOP ("Do you want perform activities of loop block?") {
    activityA;
}
```

In other situations, different from the behavior obtained with the serial LOOP command, it is necessary to perform the blocks in parallel mode. The command DOPARALLEL express this behavior, Table 5 expresses that runtime activityA activity will be offered to the analyst group, and without the need to wait for the end of activityA the question "Do you want to rerun the block?" will be issued if the answer is positive place another instance of the activity to the group analyst, and if the answer is negative, the flow goes to the end of DOPARALLEL, the semicolon (";"), which will wait for the completion of all activities contained in the block command DOPARALLEL. This behavior is also described in the standard variant thereof in parallel WCP15 [12].

**TABLE V. THE DOPARALLEL COMMAND**

```
DOPARALLEL{
    ROLE (analyst, "a comment"){
    activityA;
    }
}WHILE ("Do you want to rerun the block?");
```

### III. ROLE ANALYSIS

In this work we are interested in roles for software development process and in CSCW-Roles. Zhu and Zhou made a survey of the literature about role mechanisms and role-based systems. They identified six different kinds of Roles: Social-Roles; Modeling-Roles; CSCW-Roles; RBAC-Roles; Systems-Roles; and Agent-Roles. According to them, Social-Roles aim to describe human’s behavior in social lives. Modeling-roles aim to describe abstract mapping to a solution of a practical problem. Agent-roles aim to simulate the collaboration of agents in a system. System-roles are an expansion to
accommodate more properties than roles of access control that aim to simplify the work of system administrators. And CSCW-roles aim to support people to collaborate with the help of computers [13].

In [14], Zhu and Zhou summarized relevant publications in the classic software development life cycle that starts form project plan, follows with system requirement analysis, design, coding and maintenance, with goal to identify roles. They concluded that is possible to introduce roles into every activity of software engineering. They found roles such as Requirements Owner, System Designer, System Analyst, Validation/Verification Engineer, Logistics/Operations Engineer, Glue among Subsystems, Customer Interface, Technical Manager, Information Manager, Process Engineer, Coordinator, and Classified Ads System Engineering.

On the other hand, CSCW is an area of research that is intended to support work among people via computer [15]. In [14] there is a list of papers that discuss roles in CSCW systems. The conclusions are that the roles in CSCW are to organize access to shared artifacts and to manage the interactions between human-computer and human-human in a shared environment.

Lucas et al. investigated six languages found in the literature that represent collaboration [16]. They concluded that languages implement the role concept independent of the domain in which the language is applied. For example, the Cooperative Systems Design Language (CSDL) was designed to support the development of cooperative systems. It uses group’s rights to represent roles. And BPMN (Business Process Model and Notation), that was made to provide an easy standard visual language, implements roles through Pool or Line.

In business process, we have BPM (Business Process Management) which is focused on processes, and Social BPM has a focus on people, the last is defined as the effort to design and execute business processes collaboratively [17]. BPM uses static roles, this is done in modeling time and setting in deployment time, as Social BPM must support the definition of new roles according to the profile of new users, it is named dynamic roles.

The definition of human roles can be classified in class, profile, boundaries, interactions, authority and responsibilities. In [18] were achieved four types of responsibilities: who controls the activity; who actually carries out the work; who provides constraining advice; and who provides discretionary advice. The controller role has direct responsibility for the activity and will delegate the work to executor role, who executes activities. Constraining advisor role to list constraints, for example cost, while discretionary advisor role collaborates with past experience in a process along.

OpenUP is a version of the Unified Process with interactive and incremental approaches within a structured lifecycle for software development focusing on collaboration [19]. OpenUP organizes the roles into three categories. Basic Roles contain the classic roles of software development such as Analyst, Architect, Developer, Project Manager, Stakeholder and Tester. Deployment roles are those that support the implementation phase of the software: Course Developer, Deployment Engineer, Deployment Manager, Product Owner, Technical Writer, and Trainer. And the Environment roles are those that support the process at runtime, i.e., the Process Engineer role is responsible for ensuring that the software process is being followed, and the Tool Specialist role to provide technical support for tools that support process execution. Furthermore, OpenUP lets you specify the role of primary executors of an activity (responsible for the execution of the activity) and additional executors of an activity (support and information for task execution).

IV. DOES COLLABRDL CONFORM TO 3C MODEL?

The 3C model is a way of characterizing collaboration and was originally described by Ellis [2] and reviewed by Fuks and colleagues [1]. This model describes collaboration through three well-defined aspects (communication, cooperation and coordination) and another fourth aspect related to the first three.

Communication is the exchange of information in order to update the knowledge in some direction. It is a human need, but it can also be applied to objects, for example, sensors for environmental monitoring and robotics [20].

Cooperation is the sharing of a working session with the aim of achieving a goal [15]. It can be facilitated through the use of roles [13,14], but it can happen in a free and spontaneous way, sustained directly by specific people.

Coordination is the management of interdependencies among activities [10]. These interdependencies determine when an activity should be performed, i.e., after which activities, before which activities, or parallel to which activities. An activity consumes products from other activities and processes are examples of the application of this concept.

The awareness, which can be seen as the understanding of the activities of others in order to gain knowledge to carry out our activities [21], is a characteristic of high human dependence, but it can be better identified when making use of roles and profiling, aligned with visualization techniques [22].

The CollabRDL commands were described based on the 3C model [1] and on some workflow patterns [12]. Below we will certify that the elements of the 3C model are contemplated in CollabRDL.

A. Communication in CollabRDL

CollabRDL does not present in his description any commands to guide or encourage communication between people. Nevertheless, in the second parameter of the ROLE command you can pass information when delegating a block of activities for a group of reusers. This may raise questions and provoke the reuser to look for clarification through the coordinator, helping to initiate a communication session. Therefore, we conclude that CollabRDL weakly describes communication, leaving for the execution environment the responsibility for providing adequate means of communication, such as chat, email, discussion lists, video conferencing, and so on.

---

1 Other models exist in the related literature, such as Borges and colleagues [3], in which appears the perspective of group memory.
B. Cooperation in CollabRDL

CollabRDL is an extension of RDL, and the core of RDL is based in a building environment imperative language [23], describing step-by-step what should be done and in which order. In this paradigm we can create a running instance of a reuse process, and in the case of CollabRDL, we can share a running instance in a work session where people can contribute with their activities through the use of labels of roles. Therefore CollabRDL meets this perspective of the 3C model.

C. Coordination in CollabRDL

CollabRDL is imperative and for this reason implements the notion of order. Thus, an activity can be described after another. Furthermore, through the commands PARALLEL and DOPARALLEL we can provide activities in parallel, a key requirement for coordinating activities in the context of group work. Therefore, due to the imperative paradigm and the presence of PARALLEL and DOPARALLEL commands, CollabRDL implements the perspective of coordination according to the 3C model.

D. Awareness in CollabRDL

Awareness is a human activity that involves the need for summarization of the past knowledge, added to the present context. Moreover, it involves personal characteristics such as the human senses (sight, hearing, taste, smell, and touch), physical health (pain, pleasure, malaise) and feeling (state of happiness, sadness, indifference, depression). This indicates that the awareness of something changes over time. In this sense, awareness must be supported in the execution environment of CollabRDL. Currently CollabRDL runs in an environment of BPMN [24] focused on the execution of business processes. This type of environment was chosen because we consider the reuse process similar to a business process. We conclude therefore that CollabRDL delegates the awareness support to the execution environment of business processes.

V. IMPROVING THE COLLABORATION SUPPORT IN COLLABRDL

In general, software projects are context-dependent: they depend on people, tools, social conditions and technology [25]. In this context, some situations are difficult to predict, such as retention of people and project costs. So you need to monitor and make decisions at runtime. Accordingly, in CollabRDL, when an activity is assigned to a group, you can not predict what will be the project cost and whether this will imply, for instance, in performing or not an activity (if it is optional). Thus, the decision to perform or not an activity should be decided at runtime by consulting the group responsible for restrictions. On the other hand, for optional or alternative activities, the opinion of a technical expert with knowledge of the current project (and of the company environment) becomes relevant in decision making.

Collaboration can be seen as a group of people working together to achieve a goal [26]. The ROLE command of CollabRDL considers that the realization of a block of activities is the goal to be achieved. However, CollabRDL allows to create a block of activity with only one activity. In this case, the activity will be carried out by only one person and the communication between the controller (who delegates) and the executor is neither encouraged nor monitored.

Callan and colleagues [18] present the notion of abstract roles, i.e., not dependent on the type of project and described as follows: the controller of the activity, the executor of the activity, the constraining advisors and the discretionary advisors. In this same direction, the OpenUP tasks (activities) let you specify the Primary Performer and Additional Performers in order to encourage collaboration in performing tasks.

The command ROLE describes the roles of controller and executor of activities, but does not consider the role of advisors (or Additional Performers). Thus, it is necessary here to justify the inclusion of these new roles and discuss how they should be described in CollabRDL.

Thus, the ROLE command should allow the (reuse) process modeler to describe that, in order to carry out a block of activities, the executor must communicate with certain groups of advisors, such as constraining and discretionary advisors, technical support etc.

VI. CONCLUSION

In this article we analyzed the concept and application of roles in the literature, especially in the scope of software development process and in CSCW-Roles. We found out that role is a widely used concept in this scope. The role can be seen as an abstraction of a person, a way of classifying people, allowing us to minimize the dependence of people in a process of software reuse without compromising on quality and productivity.

We also analyzed that a process of software reuse guided by CollabRDL is in accordance with the 3C model. However, improving the ROLE command by indicating that the activity executor must communicate with certain groups of advisors is a strong indicator that will improve the support for collaboration in CollabRDL.

Future work includes investigating how we could measure collaboration in a process of software reuse guided by CollabRDL. The aim is to plan an experimental study to compare the support for collaboration in the current version of CollabRDL, presented in Section 2, with the new version proposed in this article.

ACKNOWLEDGMENT

The authors thank the CAPES and CNPq Brazilian agencies for the financial support.

REFERENCES


