An Expressive Role-based Approach for Improving Distributed Collaboration Transparency

Abdullah O. Al-Zaghameem[1] and Mohammad Alfraheed[2]

[1] Department of Mathematics and Computer Science, Tafila Technical University
Tafila, 66110, Jordan

[2] Department of Mathematics and Computer Science, Tafila Technical University
Tafila, 66110, Jordan

Abstract

This paper presents an approach for applying the Remote Role-Playing (RRP) concepts in the development of Distributed Collaborative Applications (DCAs) that use the collaboration-based (role-based) design. The paper studies a programming model called Object Teams (OT), which aims at implementing this design for the object-orientated languages. Then, it introduces the RRP for improving the DCAs modularity through mapping the fundamentals of the OT model to distributed environments. The approach is demonstrated through developing a simple case study.

Keywords: *Distributed collaborative application (DCA); DCA Modularity; Role-based design; Object Teams (OT)*

1. Introduction

A Distributed Collaborative Application (DCA) could be defined as a group of separated programs which are executed on several network nodes to achieve a shared goal. The demand to smoothly design and implement DCAs is highly increased as the complexity of the development of these applications has increased too. The main complexity in the development of DCAs is the decomposition of system functionalities into separated components [3]. Recently, the Aspect-Oriented Programming (AOP) proves itself as a promising technique to improve the quality of software by decreasing the level of code scattering and tangling [2] [4]. As a consequence, AOP concepts are applied in various software engineering fields. For instance, several approaches have been developed like Tako-approach [6] (a black-box approach) and AO-CVE - approach [1], to improve the modularity of the DCAs by separating the *collaborative functionalities (CFs)* of components from the application core functionality.

In the context of collaboration modularity, the Object-oriented Collaboration-based Design (OOCBD) describes a methodology for decomposing object-oriented applications into a set of classes and a set of collaboration modules [7]. The AOP-based approaches lack the appropriate module for expressing collaboration modules. One of the recent approaches that adopt the OOCBD is Object Teams (OT) [5]. The OT programming model offers a Role-based technique [9] [10]. Role is one of the foundation concepts for building collaborative applications [8]. .

The OT model captures collaborations in new modules called *Teams*, and separates the collaborative functionality of application objects inside modules called *Roles* [5 and 14]. An object in a collaborative application participates by playing a role in the collaboration via Role-playing process [11]. The OT model has many features; the capability to decompose collaborative-based applications in modules in expressive and modular way. Moreover, offering an integration foundation between the AOP concepts and the Object-orientation. The main contribution of this research is to map the OT's features to distributed environments. The realization of this mapping improves the DCAs modularity and offers expressive representation of the CFs; hence simplifies DCA composition. In addition, the mapping assists developers to construct easy to develop, evolve, and maintain DCAs.

In this paper, the Remote Role-Playing (RRP) is introduced as an approach to enable the distributed object-oriented application components to play the roles of "teams" remotely (i.e. in distributed environment), while preserving the semantics of the OT role-playing. The team module in RRP (called *remote team*) is used to represent the context of the collaboration, which enables the dynamic management of collaboration activities like *activating/deactivating* objects participation. The role modules (called *remote roles*), on the other hand, are used to capture the CFs of application components in collaborations. The utilization of modularity of collaboration-based design in RRP facilitates resolving primary collaborative problems like conflict resolution [12] and early conflict resolution [13]; because remote roles *intercept* the CFs of application objects before they are employed in collaboration.

The paper is organized as follows: in Section 2, a simple collaborative application will be developed as a case study. Section 3 presents an overview of the OT model, and highlights its key features. Section 4 presents the RRP approach; which maps OT features to distributed environments. The case study of Section 2 will be used to

demonstrate the RRP. The section addresses the requirements of accomplishment the mapping process and emphasizes the features offered by RRP. In Section 5, an evaluation discussion will be held for the performance of RRP. Section 6 discusses the related works, and in Section 7 the conclusions.

2. Case Study: Simple Painting Collaboration

In this paper, the following simple collaborative scenario will be used to demonstrate the distributed collaboration based on remote role-playing: Consider a collaboration of multiple users for drawing a shared "painting". Using the object-orientation model, the following primary application entities could be recognized: the painter (who performs drawing), the shapes which a painter can draw, and the place on which shapes could be drawn. In the world of patterns, the Model-View-Controller pattern fits best the design of this application. The painter (the Controller) works on a set of shapes (the Model). The controller can create and add shapes on a graphical interface (View). Fig. 1 illustrates the UML class diagram of this composition. Java programming language is used to implement the case study as shown in Fig. 2. The figure illustrates a code snippet of the implementation of **Painter** class.

Inspecting the code in Fig. 2, the method **paint** is recognized as a *Collaborative Functionality (CF)*. That is, it points out to the part of painter object's behavior in the collaboration processes. Depending on object-oriented Collaboration-based Design (CBD), the "painting collaboration" *crosscuts* painters' functionality at paint method. Fig. 3 shows the CBD, where **CollPainting** (the name of our painting collaboration) crosscuts **Painter** class.

In OT model, the intersection between **Painter** class and **CollPainting** collaboration is the *role* which **Painter** objects can play in that collaboration. The OT model captures the **CollPaining** collaboration in a *team* module. It captures the CF of **Painter** class in the team within role module. For modeling purposes, the OT model uses UML notions to model collaborations as an integration of package and class diagrams. Fig. 4 illustrates the OT representation of "collaborative painting" application. In this way, the CF of **Painter** class is captured in the **CoPainter** role.

A Team module in OT/J [14] (the programming language implementing the OT model in Java) is a first-class entity. Thus, it can declare attributes and methods (see Fig. 4), extends other teams and, most importantly, it can be instantiated. Likewise, roles can have their own attributes and methods.

The OT model organizes the collaborative relationship between teams and application classes (called *bases*) through a high expressive relationship called *playedBy*. The "playedBy" relationship selects one base class to play exactly one role in any specific team. At runtime, it binds base objects to role instances by the mediation of team instance.

2.1 The Collaborating Process

To facilitate application base objects participation in teams, the "playedBy" relationship establishes two types of communication between bases and their roles: the first one is called *Callin Method Binding (CIMB)*, which enables base objects to *call into* role instances specific methods *after*, *before*, or in *replacement* of their methods. For example, the expression {collPaint ← **after** paint;}, shown in Fig. 4, is a CIMB that instructs **Painter** objects to call the method **collPaint** *after* they call their **paint** method. The method **collPaint** is named as role's *callin* method. In fact, the OT model introduces the CIMB expression and role's callins as counterparts to, respectively, *pointcut* and *advice* concepts of AOP languages like AspectJ [15].

The second communication type is called *Callout Method Binding (COMB)*, and indicates that a role instance declare a method, which is not available locally, by *calling out* to a method of the associated base object.

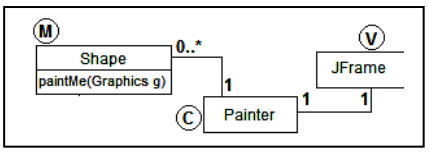

Fig1. Class diagram of painting application

```
1. class Painter
2.  implements MouseListener, ..
3. {
4.       JFrame window = ..;
5.       List<Shape> shapesList = ..;
         :
6.       public void prepareShape(Shape s){..}
7.       public void paint(Shape s) {..}
8. }
```

Fig.2 Part of **Painter** class implementation in Java.

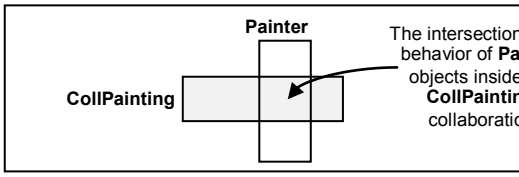

Fig.3 CBD of the collaborative Painting application

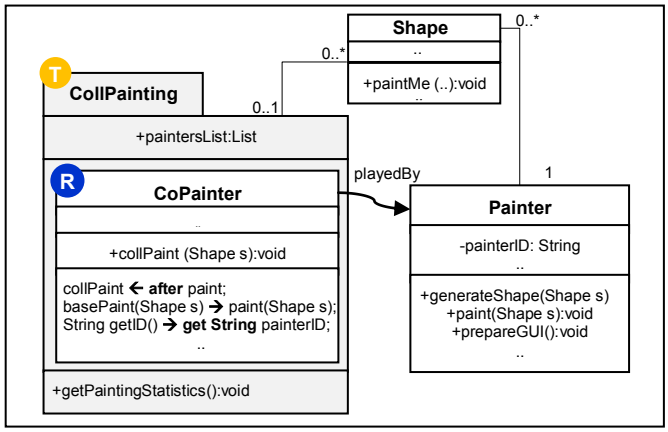

Fig.4 Modularizing the CBD of "painting collaboration" in OT.

OT presents two types of COMBs: the first is called method-COMB, which indicates that the stated role method will invoke a base method. For example, the expression {basePaint(Shape s) → paint(Shape s);} will dispatch the calls made to **basePaint** method (which is never implemented) toward the base method **paint**. The second COMB type is called field-COMB, which enables role instances to *get* or *set* the values of their base objects' fields. For example, the expression {String getID() → **get String** painterID;} will get the value of **painterID** field of the bound base object whenever a call is made to the method **getID**.

The CIMB and COMB mechanisms together form a complete communication channel between base objects and their roles, which facilitates control and data flow, and fulfills the needs of role and base objects for accomplishing precise collaborating.

2.2 Collaboration Programming

In OT/J, developers can define teams by using the keyword "`team`", and define role classes as inner-classes. Fig. 5 shows a code snippet of the implementation of **CollPainting** team. Note the expressive binding between **Painter** base class and **CoPainter** role class through the keyword "`playedBy`" at line 3.

3. Obstacles to Remote Role-Playing in OT/J

The Remote Role-Playing (RRP) is the technique aims at enabling the objects of distributed applications to play the roles of OT/J applications *remotely* while preserving the semantics of the OT role-playing. In practice, the OT/J programming language involves the following specifications that technically prevent the remote role-playing to be applied:

- Obs-1: The OT/J's *weaver* adopts the mechanism of *Load-Time Transformation (LTT)* [14] to weave roles in the bytecode of application base classes. The LTT technique is widely used in AOP for injecting aspects into the application's business logic. In general, the using of LTT results in hard coding roles' callin methods inside base classes' bytecode. This will prevent mapping OT/J applications to distributed environments because teams are taught to bind their roles to local base objects only. Similarly, role classes are compiled and transformed in a way the generated role instances deal with local base references only, which are irreplaceable by remote references at the source code.
- Obs-2: OT/J supports only *static* role-playing, i.e. a specific base object can play only those roles that woven into its base class. Thus, it cannot play new roles *dynamically* at runtime. Practically, this impacts the capability of OT/J in developing dynamic collaborations.
- Obs-3: In Java-based distributed applications, distributed components often represented via *contract-based* designs like CORBA-IDL [16] or *remote interfaces* of Java-RMI [17]. In both cases, OT/J does not fully support playing roles by bases that are *interface*; due to implementation limitations [14] (§2.1.1).

```
1. public team class CollPainting{
2.   List paintersList =…;
3.   protected class CoPainter playedBy Painter {
       ..
4.    public void collPaint(Shape s)
5.    {
6.       System.out.println("Painter:"+getID());

        /* paint the shape s at all
                other participants painters */
7.     for(CoPainter coP : paintersList)
8.      if(coP != this)
9.        coP.basePaint(s);
10.   }

11.   collPaint <- after paint;                  // CIMB
12.   basePaint(Shape s) -> paint(Shape s);      // COMB
13.   String getID() -> get String painterID;    // COMB
14. }
15. public void getPaintingStatistics() {…}
16.}
```

Fig. 5 Implementation of **CollPaining** team in OT/J.

4. Mapping OT/J Applications to Distributed Environments

In this section, the conception of mapping OT/J applications to distributed environments is presented in order to investigate the shortcomings of OT/J to support the remote playing of roles. The section first addresses the requirements for a precise mapping (see section 4.2), and introduces the Remote Role-Playing (RRP) (see section 4.3).

4.1 Distributing OT/J Applications

To make the application of case study in Section 2 a true DCA, painters need to be able to participate in "painting collaboration" over a real distributed environment. The participation imposes the existence of **Painter** objects and **CollPainting** team (and its role instances) at separate network nodes or application processes. To demonstrate the impact of technical problems described previously (in section 2), three Painter-application instances are deployed on three different nodes (H1, H2, and H3) as shown in Fig. 6. In

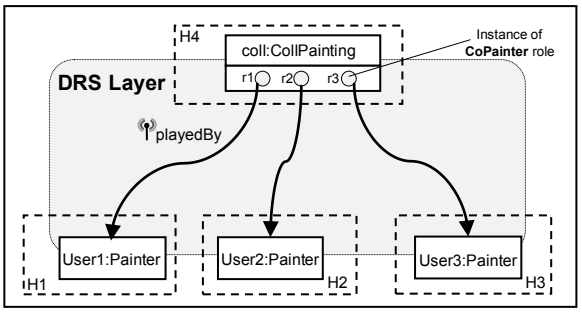

Fig.6 Deploying "Painting collaboration" on distributed Environment.

addition, an individual **CollPainting** team instance must be deployed at each node to capacitate the deployed **Painter** objects playing the **CoPainter** role. In this case, three-separated *local collaborations* (*teams*) must be created. The problem arises here is that "User1" (H1) is not be able to share the CF (i.e. painting) with "User2" (H2) and "User3" (H3) because she plays a local role.

Another problem shows up is that the state of separated local-collaboration instances (i.e. teams) must be synchronized all the time. For example, the activation of one of them should lead to activating others. The same thing must be done in case of collaboration deactivation; otherwise, collaboration inconsistency is encountered.

To overcome these problems, a single team instance has to be deployed and all **Painter** objects should play its roles remotely; thus, synchronizing team-instance copies is eliminated instead of preserving a unified consistent state among them. Furthermore, the inter-relationships between team's roles are preserved (if any) unbroken. For these reasons, the RRP is introduced as a communication layer over which the deployed **Painter** objects can play the **CoPainter** role *remotely* (see Fig. 4). The "playedBy" relationship is marked with the *antenna symbol* to indicate that it binds discrete base and role objects. Having a single team instance is very important to unify the effects of role-playing on base objects via team *activation/deactivation* operations.

4.2 Requirements of RRP

The separation of base objects and team instance of **CollPainting** results in the broken of "playedBy" relationships. The breaking results due to hard coding roles inside base class's bytecode by the OT/J transformers. The fundamental idea in the RRP approach is to *reformulate* base and team classes (including roles), which are involved in *remote* "playedBy" relationships. In such way, their objects are able to communicate over distributed environments. This communication should guarantee precise CIMB and COMB executions, and preserves the semantics of the local "playedBy" relationship. Therefore, the accomplishment of the following primary Requirements (R) is essential in order to realize that communication:

- **R1:** The participation of distinct base objects of class **Painter** in the collaboration. In this requirement, the capability of the base objects of class **Painter** deployed on hosts H1, H2 and H3 (hereafter called *remote base objects*) has to participate in the collaboration. In OT/J, this requirement implies that remote base object's functionality must be remained intercepted by the role's CIMBs. Moreover, remote base objects must determine the location of the team instance, which encloses their roles before any of their functionalities (i.e. methods) is invoked. Simultaneously, a remote base object must *dispatch* its CF remotely to the desired team instance; thus, a distributed collaboration could be achieved.
- R2: *creating* role instances based on demand. In this requirement, the capability of the **CollPainting** team instance deployed on host H4 (hereafter called *remote team instance*) has to *create* role instances on-demand. Therefore, the requirement involves its capability to bind the created role instances to the remote base objects.
- R3: addressing the declared COMBs on the exact remote base objects. In this requirement, the capability of role instances of **CoPainter** has to issue the declared COMBs on the exact remote base objects associated with. For example, the role instance "r1" (shown in Fig. 6) should invoke the field-COMB **getID**() (see lines 6 and 13 of Fig. 5) on the remote base object "User1" only.

4.3 The RRP Implementation

In terms of implementation of the RRP, two issues have to be taken in account. First, carrying out these requirements imposes the relaxation of tight coupling between base and team classes (including its *remote* roles) before they are loaded into the Java Virtual Machines (JVMs) for execution. Furthermore, requirements of RRP have to be fulfilled without violating the core functionality of base or team objects. In other words, remote base objects of **Painter** class have not to detect that they are playing remote roles, and they have to exhibit their behaviors, even though, for example, the team found deactivated. Second, the remote base objects must be affected by the role-playing to achieve the participation in the collaboration. To achieve these two issues, the (Distributed OT/J (DOT/J)) Runtime System (DRS) layer is here developed to first operate underneath the remote objects (bases and teams) and to second facilitate the RRP activities (see Fig. 6). Therefore, a need to communication layer is aroused. Two types of communications are introduced:

1) An inter-communication between the DRS components: The DRS layer comprises two main units (as shown in Fig. 7) which are the Group Communication Unit (GCU) and the Distributed Objects and Teams Manager (DOTM). The latter provides the necessary coordination for remote base objects and team instances in a distributed OT/J application to accomplish a precise binding. The Group Communication Unit (GCU), on the other hand, is responsible of the *transparent* deployment of team instances and the conteatxual information of roles on every application node. Moreover, GCU implements a reliable

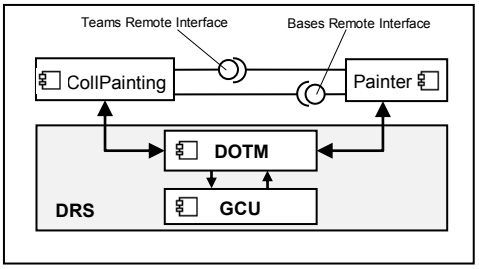

Fig.7 The RRP Runtime Infrastructure.

communication protocol between the deployed DOTMs, so that they can preserve the collaboration consistency by synchronizing their states; in particular, the *Remote Teams List (RTL)*.

2) The actual RRP communication: This type of communication involves the direct communication between remote base objects and their roles (via the remote team instance) to execute the CIMBs and COMBs declared in these roles. Establishing smoothly this type of communication requires that remote base objects and remote teams have to communicate remotely. In addition, the relaxation of base and role bytecodes needs to be accomplished. Therefore, the *Provided/Required Interface* design is adopted to glue the remote base objects and remote teams of application. The provided/required interface design nicely captures the relationship between remore objects. Thus, remote base objects of **Painter** must provide a specific remote interface, and require another interface from the remote team **CollPainting**. Likewise, the remote team instance of **CollPainting** provides a remote interface and requires the interface of **Painter** objects. Fig. 7 depicts this relationship.

In order to preserve the integrity of remote base and remote team objects, the structure of their classes after *gluing* must not be negatively violated, i.e. their hierarchical structures must remain untouched. Consequently, the LTT is adopted to *reformulate* remote base **Painter** and remote team **CollPainting** classes into *distributed components* as shown in Fig. 7. The transformation details are beyond the scope of this paper. Nevertheless, it is important here to mention that our bytecode transformers (see Obs 1 in section 3) *reuse* the transformed classes of the OT/J application, and carefully *replace* the features of local "playedBy" relationship with remote ones. For example, role instance "r1" (shown in Fig. 6) is bound to the *remote stub* generated from the remote interface provided by the remote base object "User1".

4.4 The Engagement between Remote Base Objects and Remote Team Instances

Enabling remote base objects of **Painter** class to allocate dynamically the remote team instance "coll" requires that the team instance *registers* itself in the DOTM (i.e. DOTM runs on host H4). The registration process involves providing the local DOTM component with contextual information about the team instance such as its *remote stub* and a list of the CIMB expressions declared by its *remote* roles. As a consequence, the DOTM *broadcasts* the record of the registered team (via the GCU) to all the DOTMs executed on other application nodes. Moreover, the DOTMs deployed on hosts H1, H2 and H3 keep an identical RTL.

Once a registration record arrives to the local DOTM at host H1, it *notifies* all the remote base objects coexisting at the same node (in this case the instance "User1"). At this moment, "User1" obtains the knowledge that a team instance comprises one of its roles. In OT/J, a base object starts playing a role *after* it has been bounded to an instance of that role. The binding process has been carried out when one of base object's methods has been intercepted by a declared CIMB. Thus, as "User1" already obtains the recent RTL, it can check whether any of its functionalities has matched any of the CIMBs. To facilitate this task, the bytecode transformer injects a *trap* at every declared method in **Painter** class, and dispatches their calls to a central *dispatcher* method. The dispatcher method traverses the RTL and extracts those CIMBs matching the current invocation.

Here, two different transformation strategies can be chosen:

The first is to invasively trap all base methods (called *total hook weaving* [18]); thus, the dynamic playing of roles is supported at runtime whenever new teams are attached to the application without interrupting its execution. Within this strategy, the remote team instances and remote base objects are required to provide *generic remote interfaces*. As a consequence, remote base objects can *plug* into any dynamic collaboration (team), and different remote team instances can bind the same remote base object using a single remote stub. In addition, a transparent conversion of base objects into collaborative-objects is achieved.

The second strategy is to declare those base class functionalities desired for being intercepted by CIMBs in a data file (like XML). The latter is then directed to the transformer in order to target only these functionalities. In this way, a great performance is saved overhead at runtime. However, the desired dynamic evolution of the DCAs is restricted.

When "User1" draws a specific shape, the method **paint** is called and trapped by the dispatcher. The dispatcher, in turn, detects a CIMB matching. Consequently, the dispatcher extracts the accurate *Team Level Wrapper Method (TLWM)* of the remote team instance "coll" from the contextual information. In addition, the extraction guarantees that the TLWM corresponds to the role's call in method **collPaint**. The latter then invokes that TLWM via the remote stub of "coll". The team instance "coll" receives, as an argument to TLWM, the remote stub of "User1" base object. Therefore, it can create a new role instance (in our case it was "r1") and binds it to the remote stub. According to the latter, the role instance "r1" can address COMBs on the remote base object "User1" using the RMI invocations.

5. Evaluation

To evaluate RRP, the application shown in Fig. 6 is executed as follows: first, the Team-application is run at host H4. Therefore the three Painter-applications are executed at H1, H2 and H3. The role **CoPainter is** implemented, so that it adds the participant painter to the painters list (see line 2 of Fig. 5). We achieve this by intercepting the method **prepareGUI** of **Painter** objects

by a CIMB (not shown in Fig. 5). After then, each one of the painters "User1", "User2", and "User3" paints randomly 10 shapes. The **CollPainting** team has been provided with a GUI, so it can clone the *shared painting* as well. The role's callin method collPaint will take care of synchronizing the painted shapes of a specific painter at others GUI (lines 7 to 9 of Fig. 5). As expected, the 30 shapes appear on all Painter- and Team-applications' GUIs.

For performance analysis, the runtime required for carrying out a complete CIMB interceptions is recorded , which includes (1) dispatching the control flow from remote base object to the bounded role instance, (2) broadcasting the painted shape to all other painters, and (3) painting that shape at the team's GUI. The recorded values (in milliseconds) are shown in Table 1. The average runtime values give clear symptoms for a promising approach. However, further evaluation on more advanced case studies is one of our future works. The time required at the first interception in all cases is higher than the rest. The extra time is consumed by the Java-RMI system for generating dynamic proxy objects.

Table 1: Runtime values for intercepting **paint** method 10 times at the three painter-applications.

No.	User1 (H1)	User2 (H2)	User3 (H3)	Avg.
1	50	20	30	**25**
2	16	14	7	**10**
3	10	13	7	**8**
4	12	6	6	**7**
5	14	13	6	**10**
6	17	16	10	**12**
7	8	15	7	**9**
8	18	13	10	**12**
9	11	9	11	**10**
10	17	13	6	**12**
Avg.	**17.3**	**13.2**	**10**	**11.5**

6. Related Works

The RRP could be considered as a distributed-AOP approach; because it supports intercepting the remote base objects' functionalities by the remote CIMBs of roles, and then executing the associated advices *remotely*. In addition, it could be introduced as an approach for improving the composition and modularity of DCAs. In this regard, the RRP shares several features with TaKo [6]. First, RRP and TaKo are AOP-approaches which address the transparent collaboration between legacy applications that were not designed for collaborating. TaKo proposes a full blackbox approach for supporting collaboration transparency. However, it is environment-specific as it targets AWT- and Swing-based Java applications only. In contrast, RRP presents more expressive approach to design and model the collaboration and the collaborating functionalities. After that, the RRP infrastructure operates to provide the necessary facilitation for accurate distributed collaboration. In RRP, the strategies of solving problems like Collision Detection, Priority, Logging, etc. are easy to customize and implement.

In [1], a proposal for employing the AOP concepts in the development of collaborative virtual environments (CVEs) is presented. The approach stands on intercepting the functionalities of application components, and dispatch control flow to a dedicated middleware layer, which interconnects between components and aspects dynamically. All aspects must be previously registered in this layer to ensure accurate interconnections. The RRP offers a similar approach; remote team instances need to register in the DOTM. However, the AO-CVE as an AOP-based approach lacks support for clear collaboration modularity and expressive relationships between collaborative components and aspects as in RRP.

Service-Oriented Architecture (SOA) is used in [19] besides a set of off-the-shelf technologies to develop a collaborative authoring application, which involves five users collaborating over the internet. This and other approaches like CoDesign [13] and GroupUML [20] are classified as collaborative software design and modeling environments. These and alike approaches cope mainly with problems of conflict detection, shared state and time synchronization, among others. Our approach offers better separation between the core functionality of application components and the collaborative functionalities each component exhibits in the collaboration. Even though SOA is (in concept) similar to team module of OT, teams provide twofold facilities; a clear contextual collaborations and providing services. In addition, the client-server topology depicted in base-team relationships capacitates team instances for resolving any CFs conflicts.

7. Conclusions and Future work

This paper has presented the Remote Role-Playing (RRP) concept as a promising approach for improving the modularity and composition of distributed collaborative applications (DCAs). The realization of this approach has been discussed through the process of mapping the fundamentals of the Object Teams (OT) model. The OT model implements the collaboration-based (role-based) designs for the object-oriented languages. A primary key feature of the OT model is that it presents a high expressive modularization technique through introducing collaborations as first-class entities; which makes RRP an approach that is neither total black-box nor total white-box.

The paper has presented a simple case study to demonstrate the approach and emphasize its capability for supporting transparent collaboration between legacy Java-based applications without their source code needed. In addition, it can offer several programming capabilities for DCAs like managing collaboration contexts via team instances, besides the dynamic participation in collaborations. That is, applications' base objects can enroll in several collaborations dynamically. As future work, further evaluations need to be established in real case studies.

References

[1] M. Pinto, M. Amor, L. Fuentes, and J. M. Troya., "Collaborative virtual environment development: An Aspect-oriented approach," In Proceedings of DDMA Workshop. Phoenix, Arizona, pp. 97-102, April 2001.

[2] G. Kiczales, and *et al.*, "Aspect-oriented Programming," In European Conference on Object-Oriented Programming - ECOOP'97, 1997.

[3] M. Pinto, L. Fuentes, J.M. Troya, "Towards an aspect-oriented framework in the design of collaborative virtual environments," In Proceedings of the Eighth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS.01), 2001.

[4] S. Subotic, and J. Bishop, "Emergent Behaviour of Aspects in High Performance and Distributed Computing," Proceedings of SAICSIT'05, pp. 11–19, 2005.

[5] S. Herrmann, "Object Teams: Improving Modularity for Crosscutting Collaborations," In NetObjectDays Conference on Objects, Components, Architectures, Services, and Applications for a Networked World (NODe '02), Springer-Verlag, London, UK, pp. 248-264, 2002.

[6] R. Mondéjar, P. García-López, E. Fernández-Casado, and C. Pairot., "TaKo: Providing transparent collaboration on single-user applications," Computer Languages, Systems and Structures, Elsevier Science Publishers, Amsterdam, The Netherlands, Vol. 38, Issue. 1, pp. 108-121, April 2012.

[7] M. Mezini and K Lieberherr, "Adaptive plug-and-play components for evolutionary software development," In Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '98). ACM, New York, NY, USA, pp. 97-116, October 1998.

[8] H. Zhu, "Some issues of role-based collaboration," Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on , vol.2, pp. 687- 690, May 2003.

[9] H. Zhu, "Role mechanisms in collaborative systems," International Journal of Production Research, Vol. 44, No. 1, pp. 181–193, January 2006.

[10] H. Zhu and M.C. Zhou, "Role-based collaboration and its kernel mechanisms," IEEE Transactions on Systems, MAN, and Cybernetics —Part C: Applications and Reviews, vol. 36, no. 4, pp. 578-589, July 2006.

[11] M. VanHilst and D. Notkin, "Using role components in implement collaboration-based designs," In Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '96). ACM, New York, NY, USA, vol. 31, no. 10, pp.359-369, October 1996.

[12] H. Zhu, "A Role-based conflict resolution method for a collaborative system," IEEE International Conference on Systems, Man and Cybernetics, vol. 5, pp. 4135-4140, October 2003.

[13] J. Y. Bang and et al., "CoDesign: a highly extensible collaborative software modeling framework," In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - (ICSE '10), ACM, New York, NY, USA, vol. 2, pp. 243-246, 2010.

[14] S. Herrmann, C. Hundt, and M. Mosconi, "OT/J language definition v1.3," Technical Universitäte Berlin. Object Teams home-page: http://www.objectteams.org, (2009), last visited: March 2012.

[15] G. Kiczales and et al., "An overview of AspectJ," In Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP '01), Springer-Verlag, London, UK, pp. 327-353, 2001.

[16] Object Management Group (OMG), CORBA v3.1 , Release Date: January 2008, Home-page: http://www.omg.org/spec/CORBA/3.1/, last visited: March 2012.

[17] Sun Microsystems. "Java Remote Method Invocation Specification v1.5.0," Sun Microsystems Inc., Santa Clara, California,U.S.A. 2004.

[18] R. Chitchyan and I. Sommerville, "Comparing dynamic AO systems," In Proceedings of the AOSD'04 - dynamic aspects workshop. Lancaster, UK, pp. 23-36, 2004.

[19] A. Roczniak, J. Melhem, P. L´evy and A. El-Saddik, "Design of distributed collaborative application through service aggregation," In Proceedings of the 10th IEEE international symposium on Distributed Simulation and Real-Time Applications (DS-RT '06). IEEE Computer Society, Washington, DC, USA, pp.165-174, 2006.

[20] N. Boulila, "Group support for distributed collaborative concurrent software modeling," In Proceedings of the 19th IEEE international conference on Automated software engineering (ASE '04). IEEE Computer Society, Washington, DC, USA, pp. 422-425, 2004.

Abdullah O. Al-Zaghameem (Ph.D) a lecturer in Tafila Technical University (Tafila, Jordan) since September, 1999. He was awarded the doctoral degree of Computer Science in Software Engineering from Technical University of Berlin (Germany) in September, 2012. His current research interests include: distributed AOP, collaboration-based computing, and Information Systems Design.

Mohammad Alfraheed (Ph.D) a lecturer in Tafila Technical University (Tafila, Jordan) since September, 2012. He has the doctoral degree of Computer Science in Computer Vision from RWTH-Aachen University (Germany) in July, 2012. His current research interests include: Digital Image Processing, Computer Vision, Robotics and Software Engineering.