A Collaboration Middleware for Service Scalability in Peer-to-Peer Systems

Sung-Soo Kim, Chunglae Cho and Jongho Won
Electronics and Telecommunications Research Institute (ETRI)
Daejeon, South Korea
{*sungsoo, clcho, jhwon*}@*etri.re.kr*

***Abstract*—We introduce a novel mobile middleware which provides a collaboration service among associated apps in a symmetric fashion. This paper focuses on the challenge that how users can receive the seamless collaboration services regardless of the changes of physical device configurations in the multiscreen environment. In order to solve this problem, we propose a novel system architecture which supports primitive operations for collaborating among distributed applications, such as remote invocation, session join, session invitation, push migration, pull migration and synchronization. Our system can provide communication transparency, seamless collaboration services and scalability among heterogeneous distributed applications. The experimental results demonstrate that our system can be successfully applied to the collaboration services among multiple apps in the home network environment.**

***Keywords*-middleware, distributed applications, service scalability, collaboration services**

I. Introduction

In the fields of broadcasting especially IPTV and content delivery, multiscreen video describes video content transformed into multiple formats, bit rates and resolutions for display on smart devices such as television, mobile phone, tablet computer and computer [12]. However, previous methods related to multiscreen IPTV focused on transcoding the multimedia for adaptive content delivery. In recent years, there has been an increased interest in smart applications (or *apps*) running on various smart devices, such as smartphones and tablets. The main reason for this has been the realization that many diverse apps need a dedicated middleware for managing apps and collaborating among multiple associated apps.

A *second screen* is a second smart device used by television viewers to connect to a program they're watching [15]. A second screen is often a smartphone or tablet, where a special complementary app may allow the viewer to interact with a television program in a different way – the tablet or smartphone becomes a TV companion device. Likewise, *N-Screen* is described as a unified entertainment experience across several devices, meaning that one can flit between watching the same program on one's TV, tablet or smartphone, with the software adapting the programming to the various formats automatically. However, previous approaches related to *n*-screen services provided asymmetric collaboration based on a dedicated smart device, such as smart TV [1].

In this paper, we propose a novel mobile middleware to support the *seamless* and *symmetric* collaboration services among heterogenous multiple apps in multiscreen environments. To provide convincing collaboration services in multiscreen environments, we describe the key requirements as follows:

- *Service Discovery*: Since the wireless hosts in the wireless network are highly dynamic, service hosts should periodically announce their presence in the network. Service discovery is one of the most important functions in order to collaborate among apps through remote invocation and session join/invitation in mobile computing environments.
- *Collaboration*: The system must support collaboration services among apps in smart devices. To satisfy this requirement, the system allows apps to join or leave a *collaboration session* which is a logical space being synchronized via associated information at runtime. However, collaboration session management in a peer-to-peer environment is quite challenging since its dynamic property.
- *Mobility*: The system must support *app migration* function, which migrates certain running apps from arbitrary device to other device at runtime.
- *Scalability*: The system must be able to scale with the growth in the number of apps for collaboration services. For example, it must support the dynamic session joins for new apps during a certain collaboration service is provided. So, it is necessary to develop the collaboration session management techniques to provide the service scalability.

Main contributions: We present a novel system architecture for the multiscreen-based collaboration services, which utilizes the collaboration middleware for multiple associated apps in the same wireless network and the remote service cloud. The contributions of our work can be summarized as follows.

- *Collaboration middleware*: Our proposed middleware, which is based on peer-to-peer communication model, provides common APIs for diverse collaboration-based applications. Our middleware APIs include the primitive functions, such as, remote execution, session join/invitation and app migration.
- *Service scalability*: We introduce the schema for representing inter-app relationship in collaboration sessions. This model can help design collaboration services and add additional apps for service extension.
- *Lifecycle managment*: In order to provide the seamless

collaboration services to the users, it is important to manage the lifecycle of collaborated applications. So, we propose a method for logical app lifecycle management to support seamless collaboration session.

The rest of the paper is organized as follows. We briefly survey previous work on multiscreen middleware, multiscreen IPTV, and second screen in Section II. Section III describes the proposed system architecture and the core components in our system. Section IV presents the implementation details of the proposed middleware. In Section V, we presents the experimental results of our work in terms of service discovery. Finally, we discuss future work and conclude in Section VI.

II. Related Work

In this section, we give a brief overview of related work on service orchestration, middleware for multiscreen services, multiscreen IPTV and second screen.

Service Orchestration: Numerous approaches have been proposed for expressing service composition and orchestration in service-oriented computing [5], [9]. Lapadula et al. introduced COWS (Calculus for Orchestration of Web Services) [10], a process calculus for Web service orchestration. For isolating interactions between partners, COWS uses message correlation, the approach of WS-BPEL. Vieira et al. proposed a process-calculus model for expressing and analyzing service-based systems [16]. They introduced a model for service-oriented computation, building on the identification of some general aspects of service-based systems. Recently, SmartComposition [8] that is a component-based approach for developing multi-screen mashups was introduced. However, this approach doesn't provide *application migration* function that is one of the fundamental functions for seamless collaboration services in multiscreen environments.

Multiscreen Middleware: The potential of ambient intelligence (Aml) at home has been the subject of research for at least a decade in some major industries [2] . The goal of mobile middleware is to provide abstractions that reduce development effort, to offer programming paradigms that make developing powerful mobile applications easier, and to foster interoperability between applications [6]. The multiscreen middleware provides developers with a set of APIs and tools that optimize multiscreen experiences for various applications, such as, games, media sharing, social collaboration, and so on. A multiscreen app based on this SDK provides separate views that are connected and running on different devices [1]. The smart TV version displays a public view of the app that can be enjoyed by an audience. Mobile devices display a private view for individuals or can be used to control the action on the SmartTV. All devices (and the TV) are connected, and can communicate with each other. However, this product supports *asymmetric collaboration* since this middleware requires a smart TV as a master in centralized client/server architecture to provide a collaboration service among associated apps similar to [13].

Multiscreen IPTV and Second Screen: Many content delivery platforms are developed in order to provide adaptive content according to screen properties of smart devices [3] . These products focus on several considerations of multiscreen services in the context of over the top for multimedia content delivery. Prior researches investigated the changing television watching practices amongst early adopters of personal hard-disk video recorders and Internet downloading of video [4]. However, previous methods focused on *transcoding* and *streaming* the multimedia for adaptive content delivery according to the screen properties of smart devices [14]. A cloud-based, multiscreen, social TV system can enrich content consumption and the TV viewing experience by incorporating and displaying geolocation-aware social data for users on a second screen [11]. According to their survey in [15], email and social media usage accounted for 20% of second screen interaction, and 25% of all second screen interaction focused on communication or information retrieval specifically about the show. The trend of users integrating second screen behaviours in their viewing habits, and practitioners interest in designing systems to support them has evolved a strong research agenda. However, previous researches focused on improving the interaction model in terms of second screen services rather than collaboration services.

III. System Architecture

In this section, we describe the proposed system architecture for the multiscreen-based collaboration services.

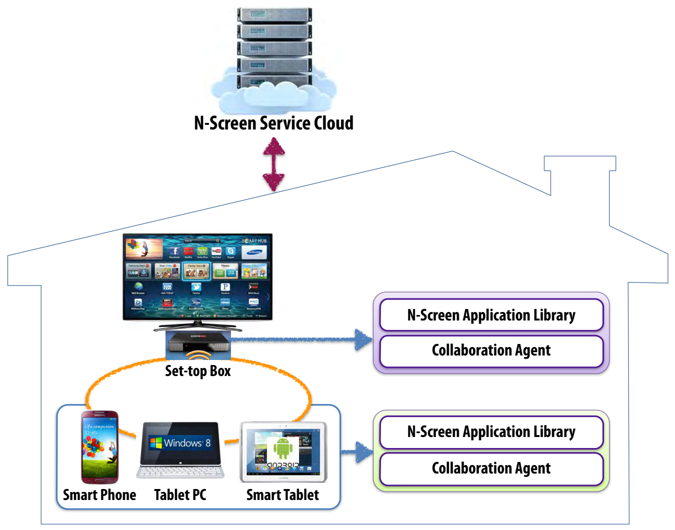

Fig. 1. Our system architecture

Our architecture consists of two major systems such as the proposed *middleware* for smart devices and *n-screen service cloud* as shown in Fig. 1. More specifically, our middleware decomposes into 2 layers; the n-screen application library (NSAL) and the collaboration agent (CA). CA is a *software agent* that provides primitive collaboration functions for each

n-screen device. NSAL is a *software library* that supports common APIs to develop a collaboration application.

A. System Overview

First, our proposed architecture based on the *Smart Home* concept is targeted at the smart home environment in terms of *communication* and *socialization*. In our work, the smart home consists of various wireless hosts such as smartphone, smart set-top, smart tablet, or laptop which connects each other through a wireless communication link in the same wireless network environment. Each wireless host has the *n-screen application library* and *collaboration agent*, which provide abstractions that reduce development effort and support interoperability between applications. These mobile devices communicate directly with each other in a *peer-to-peer* fashion with no centralized control. Hence, the *synchronization* for collaboration services is achieved through direct communication between applications.

Second, the n-screen service cloud is responsible for providing the collaboration applications similar to the apps in the *App Store* and managing the application contexts for users. The primary benefit from the n-screen service cloud is *scalability*, *persistency* and *mobility* for collaboration services. To represent a context of an application, we use the *key-value pair* which is the most simple context model. Context information of applications and collaboration sessions are stored in the context repository at the service cloud.

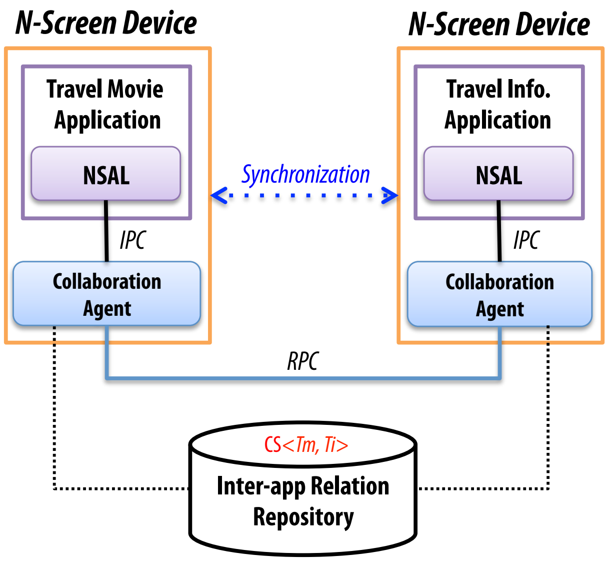

Fig. 2. An example of n-screen collaboration service. T_m denotes a travel movie application. T_i denotes a travel information application. While consuming a travel movie on smart TV through T_m, the user wants to see associated content about the currently watched travel movie through T_i. This associated content could be a related article on Wikipedia, maps, attractions information and so on.

Here, we define major terminologies for our system architecture. The n-screen device ND is a wireless host which includes the NSAL and the collaboration agent as shown in Fig. 2. NSAL communicates with collaboration agent through local *inter-process communication* (IPC) in each ND. Every ND in the same network periodically sends UDP-based broadcast message for advertisement of their aliveness. The *Remote Procedure Call* (RPC) middleware is a popular paradigm for implementing the client-server model of distributed computing. In order to collaborate among NDs, each ND uses RPC. We exploit Android Interface Definition Language (AIDL) to define the programming interface that both the client and service agree upon in order to communicate with each other using inter-process communication.

N-Screen-based Applications: There are two kinds of application; *physical* and *logical* applications (or *app*). First, the *physical application*, A_p is an application running on each device in the same network. This lifecycle of A_p is the same as the Android application and activity lifecycle. On the other hand, the *logical application*, A_l means an physical device-independent application at runtime regardless of physical device changing through application migration.

The *lifecycle manager* plays an important role for management of logical application lifecycle in our system. Even if a user migrates an application from one device to other device, the system should provide the application service seamlessly and consistently. So, our system manages the logical application lifecycle to maintain the execution states of application with contexts. Fig. 3 represents the transitions between logical application execution states. There are four states in logical application lifecycle such as *running*, *paused*, *migrating* and *stopped*. The system calls lifecycle callback methods according to state transition. This methods includes onAppStart, onAppPause, onAppResume, onAppMigrationStart, onAppMigrationFinish and onAppStop as shown in Fig. 3. Developers, who use the NSAL, should implement the onAppStart method to perform basic application startup logic that should happen only once for the entire life of the logical application. For example, their implementation of onAppStart should define the user interface and possibly instantiate some class-scope variables.

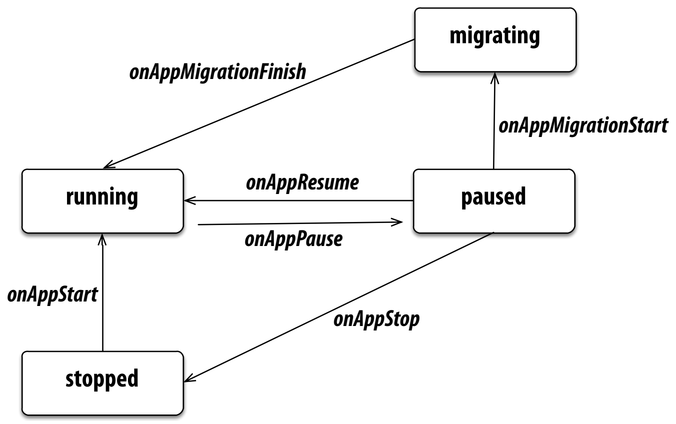

Fig. 3. State transition diagram of logical application lifecycle

Collaboration sessions: Analogy to a social community, the *collaboration session*, $\mathcal{CS}$ is defined as a logical space that can be synchronized the associated information through the more than one physical applications at runtime.

In order to represent a collaboration session, we exploit a graph-based representation $G(V, E)$, where V means a logical application set and E denotes a communication link set between two applications. An edge $e \in E$ is undirected and joins two vertices $v, u \in V$, denoted by (u, v) or (v, u). For instance, when each application like travel movie app or travel information app in Fig. 2 starts, the CA assigns a unique collaboration session ID and logical app ID for each application. If primitive collaboration operations such as session join and leave is processed, the CA will update the $\mathcal{CS}$. Fig. 2 shows the $\mathcal{CS}$ instance which consists of travel move app and travel information app. This $\mathcal{CS}$ will be destroyed when all logical app in the same collaboration session is stopped at runtime. However, a user can save a $\mathcal{CS}$ as a persistent object to the n-screen service cloud during the runtime.

Inter-app relationship: Decoupling the collaboration information from the multiscreen applications requires representing the inter-application (or *inter-app*) relationship to support the scalable collaboration services. The CA includes a manager for describing the inter-app relation among the associated apps called *inter-app relation manager* as shown in Fig. 5. This manager uses an XML-based language to encode the necessary information for discovering, executing and collaborating among the associated apps.

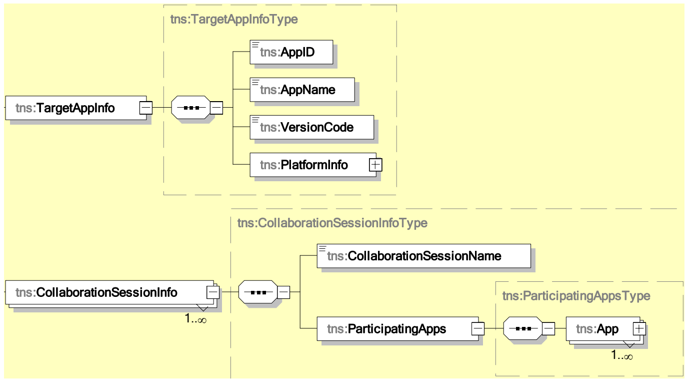

Fig. 4. The schema diagram for inter-app relationship

Fig. 4 shows the schema diagram for representing an inter-app relationship. We can describe the information of n-screen collaboration apps (TargetAppInfo) and their collaboration sessions (CollaborationSessionInfo). TargetAppInfo includes application ID, application name, version code and platform information which describe minimum hardware requirements for execution. CollaborationSessionInfo includes name of collaboration session and application information which application can participate in a collaboration session. This inter-app relationship information is stored on the n-screen service cloud. The inter-app relation manager in the CA periodically updates the information on logical storage through the RESTful API. A major benefit of the inter-app relation manager is providing the *service scalability* in terms of the collaboration.

B. N-Screen Application Library

The n-screen application library (NSAL) is the library which provides common APIs for developing multiscreen-based collaboration application. The developers can implement an application through this interface of the NSAL. Each ND can have one CA and multiple NSAL-based applications. The NSAL provides the following functions:

- *Lifecycle event handling:* In order to support migration functions, the NSAL manages the lifecycle of the NSAL-derived objects which are inherited from the NSALActivity and the NSALApplication objects.
- *Describing the inter-app relation:* The developers can describe the inter-app relationship information for connecting the relations among the associated apps. This inter-app relation is represented by the XML-based schema. If the developers want to add an additional apps for extending the collaboration, they can insert information of the apps into the inter-app relation XML file.
- *Proxy for the CA interface:* The collaboration services which are based on the NSAL APIs can run by using the primitive operations in the CA. This operations can be obtained through the CA interface.

Since the collaboration management among the apps is handled by the CA, n-screen apps are provided transparent access to a set of primitive services from the CA, thus successfully reducing the management complexity from them.

C. Collaboration Agent

The collaboration agent (CA) is a software component that acts for users or a NSAL-based apps in n-screen service environments. Each n-screen device includes a CA as a singleton process. The CA consists of nine managers which provides functions such as device discovery, lifecycle management for n-screen apps, collaboration session management, messaging, and so on as shown in Fig. 5. The *context manager* is responsible of managing contexts which are defined at n-screen applications. The *migration manager* handles the app migrations such as pull migration and push migration. The *smart device manager* deals with static (e.g. CPU and memory spec.) and dynamic profile (e.g. the amount of CPU and memory usages) of n-screen devices. The *life cycle manager* offers life cycle management for logical n-screen apps at runtime. And the *collaboration session manager* is responsible of maintaining the n-screen collaboration sessions. *Persistent collaboration session* is a reusable collaboration session information at certain time, which is stored at the n-screen service cloud. Recovering a certain collaboration session is handled by the *collaboration session recovery manager*. Each manager in the CA is *singleton object* in a n-screen device.

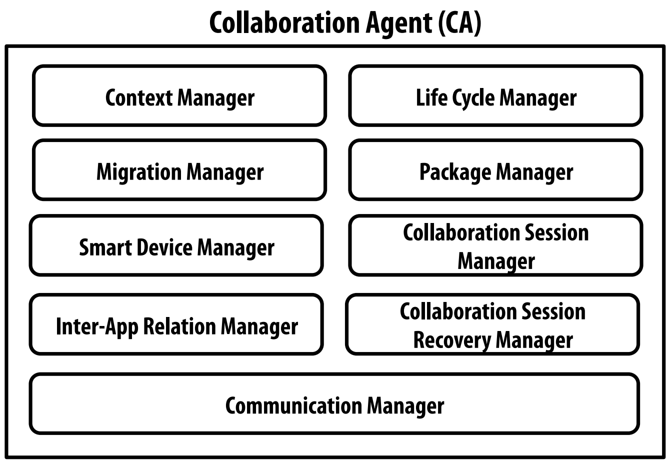

Fig. 5. The CA block architecture

Fundamental operations: The CA provides the following key operations for multiscreen-based collaboration services as shown in Fig. 6.

- *Device discovery:* From the client point of view in our system environments, device discovery allows to discover dynamically n-screen devices present in the same network. The basic interactions among n-screen devices are *service advertisement* and *service discovery*. First, service advertisement allows n-screen devices to periodically announce their presence via the UDP-based broadcast after they enter the network. And then the CA provides the service discovery function via multicast messages in order to discover the n-screen devices in the network.
- *Remote execution:* User can execute the certain n-screen app residing in other n-screen devices in the same network using a source device. For example, if a user want to execute the travel video app in remote set-top, a user can execute the remote travel video app via remote invocation using user's tablet or smartphone. This function is useful to control diverse devices effectively.
- *Session join/invitation:* In order to make the collaboration among the associated apps, user can construct the collaboration session similar to social community. One of the associated apps in the collaboration session is allowed to join or leave the session. Moreover, user can invite the apps which are not in the collaboration session using the invitation function in order to collaborate and synchronize.
- *Application migration:* The CA provides the function which migrates the running apps from arbitrary device to other device at runtime. In our work, we exploit the migration function based on the strong mobility. Our system supports two types of app migrations; *push migration* and *pull migration*. Push migration is defined as source-initiated migration. In contrast, pull migration is destination-initiated migration.
- *Synchronization:* Events and messages exchanged dynamically among all apps in the same collaboration session. We use the TCP-based multicast messages for synchronization.

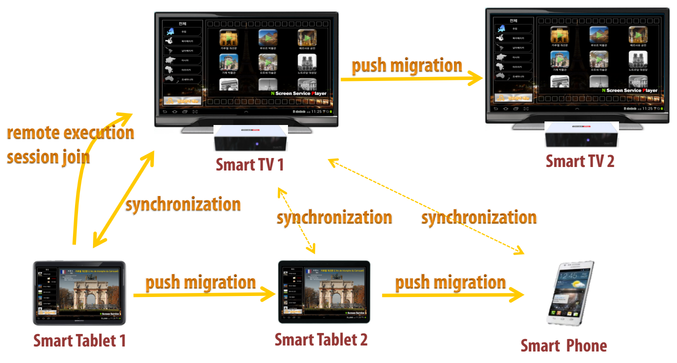

Fig. 6. Fundamental operations for collaboration services. This operations include device *discovery, remote excution, session join/invitation, application migration* and *synchronization*.

D. Collaboration Sessions

Communication plays an important role as one of the essential elements in peer-to-peer systems. There are numerous research efforts in session-based primitives for structured communication-centred programming [7], [17].

The collaboration sessions are roughly analogous to social organizations. The key approach to collaborating among the NSAL-based apps to organize several interoperable applications into a group; we call this group a *collaboration session*. The purpose of introducing collaboration sessions is to allow certain n-screen app to collaborate with collections of other smart apps as a single abstraction. Let ND_i be a i-th n-screen device in the same network, which has one or more n-screen apps and a CA. Each ND has more than one n-screen apps, A_i, which are developed by using the NSAL APIs.

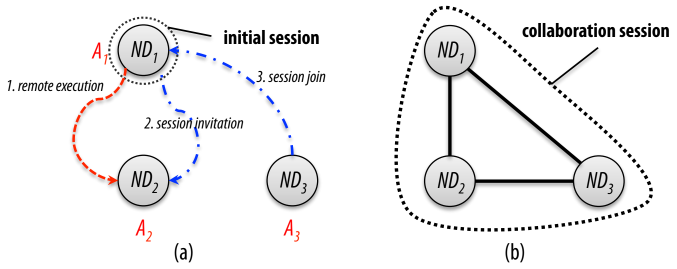

Fig. 7. An example workflow for collaboration session construction

Collaboration session construction: Fig. 7 shows an example workflow for collaboration session construction. Consider each n-screen device (ND_1, ND_2, ND_3) exists in the same network environment. First, user executes an n-screen app, A_1, in the ND_1. If the app is successfully started, NSAL creates an initial collaboration session as depicted in Fig. 7(a). In order to perform the *remote execution*, A_1 requests the service discovery function to CA. This service discovery function can be called with service properties as parameters, such as, n-screen device or n-screen apps. Then A_1 invites the other app, A_2, in ND_2 through *session invitation* function. CA updates the collaboration session after session invitation is completed. If another app, A_3, in ND_3 want to join the collaboration session by using *session join* function, CA adds A_3's logical app information to the collaboration session. Finally, the collaboration session, which includes A_1, A_2, and A_3, is constructed as shown in Fig. 7(b). The pseudo code for pipeline of this collaboration session construction is shown in **Algorithm 1**. As described earlier in Section III-C, the *collaboration session recovery manager* is responsible of recovering a collaboration session from persistent collaboration session. Fig. 8 shows an implementation of a method that recovers a collaboration session from a persistent collaboration session. CSRecoveryPerformer is an object that performs collaboration session recovery using persistent collaboration session with transaction ID.

Algorithm 1 Collaboration session construction.

```
1: procedure ConstructSession
2:     CollaborationSession $\mathcal{CS}$ ← startApp($A_1$);
3:     $A_1$.remoteExecution($ND_2$, $A_2$);
4:     $A_1$.sessionInvitation($A_2$);
5:     $\mathcal{CS}$.add($A_2$);
6:     $A_3$.sessionJoin($\mathcal{CS}$);
7:     $\mathcal{CS}$.add($A_3$);
8:     return $\mathcal{CS}$
9: end procedure
```

```java
public class CollaborationSessionRecoveryManager {
 public boolean invokeCollaborationSession
               (String tID, CollaborationSession session) {
   try {
     CSRecoveryPerformer CSRP = new CSRecoveryPerformer();
     return CSRP.invokeCollaborationSessionRecovery(tID, session);
   } catch (Exception e) {
     e.printStackTrace();
   }
     return false;
   }
}
```

Fig. 8. Implementation of collaboration session recovery

Seamless collaboration session: In our work, the multiscreen collaboration service consists of two apps; n-screen travel movie app and n-screen travel info app. The n-screen travel movie app provides guided movies related to attractions. The n-screen travel info provides various information of attractions, such as, basic information, POI data and maps. Fig. 9 shows an example of processing workflow for seamless collaboration session maintenance. First, a user constructs initial collaboration session at time t_0, $\mathcal{CS}(t_0)$, using the collaboration operations in the CA. Then user requests push migration of n-screen travel info app from a smart tablet to a smartphone, the CA at the smart tablet sends the request message, which consists of app information with app context, to the CA in the smartphone. If this migration is done, the CA will update the collaboration session at time t_1, $\mathcal{CS}(t_1)$. Finally, the physical n-screen travel app in the smart tablet leaves from collaboration session and stop the app.

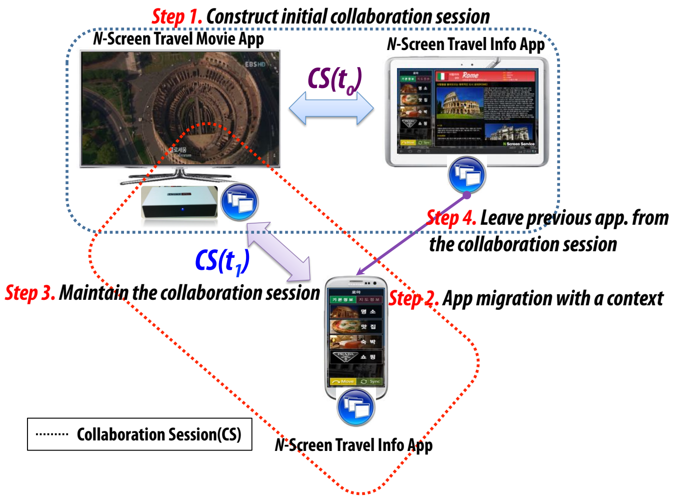
Step 1. Construct initial collaboration session
N-Screen Travel Movie App
N-Screen Travel Info App
CS(t0)
Step 4. Leave previous app. from the collaboration session
CS(t1)
Step 3. Maintain the collaboration session
Step 2. App migration with a context
Collaboration Session(CS)
N-Screen Travel Info App

Fig. 9. Seamless collaboration session: *This figure shows an example of processing workflow for seamless collaboration session maintenance.*

IV. Implementation Details

This section presents the implementation details of the proposed system. To illustrate the effectiveness of the proposed middleware, we have implemented the CA and the NSAL in Java 1.7 on Android 4.1. We focus on two aspects of the implementation: *service interface* of the collaboration agent, and *design pattern*s that we have applied for our implementation.

```java
interface CAService
{
// Remote Execution
List<SmartDevice> getNScreenDeviceList(void);
List<SmartDevice> getNScreenDeviceList(appPkgID);
List<NScreenAppPackage> getNScreenAppList(void);
List<NScreenAppPackage> getNScreenAppList(deviceID);
boolean invokeApplication(targetDeviceID, appPkgID);

// Session Join
List<NScreenSession> getNScreenSessionList(appPkgID);
NScreenAppPackage getNScreenAppPackage(logicalAppID);
SmartDevice getSmartDevice(deviceID);
String getDeviceUserFriendlyName(deviceID);
void setDeviceUserFriendlyName(deviceID, ufName);
boolean invokeSessionJoin(sessionID,  appPkgID);

// Session Invitation
NScreenSession getNScreenSessionByAppPkgID(appPkgID);
String getNScreenSessionNameByAppPkgID(appPkgID);
List<NScreenLogicalApp> getInvitationAppList(sessionName);
NScreenAppPackage getNScreenAppPackage(logicalAppID);
SmartDevice getSmartDevice(deviceID);
String getDeviceUserFriendlyName(deviceID);
void setDeviceUserFriendlyName(deviceID, ufName);
boolean invokeSessionInvitation(logicalAppID, sessionID);
void confirmSessionInvitation(bConfirm, deviceID, appPkgID);

// Push Migration
List<SmartDevice> getNScreenDeviceList(appPkgID);
String getDeviceUserFriendlyName(deviceID);
void setDeviceUserFriendlyName(deviceID, ufName);
boolean invokePushMigration(targetDeviceID, appPkgID);

// Pull Migration
List<SmartDevice> getNScreenDeviceList(void);
List<NScreenAppPackage> getRunningAppList(deviceID);
boolean invokePullMigration(targetDeviceID, appPkgID);
void confirmPullMigration(bConfirm, deviceID, appPkgID);
}
```

Fig. 10. Service interface of the collaboration agent

Service Interface: The service interface of the collaboration agent, CAService, is shown in Fig. 10. This interface includes the major methods of the CA such as remote execution, session join/invitation, push migration and pull migration. Based on this interface, we have implemented CollaborationAgentService class. This class provides a simple, high-level interface to a set of classes in the CA.

Design Patterns: First, we apply the *facade pattern* to implement CollaborationAgentService class. And, we apply the *singleton pattern* with *double-checked locking pattern* to only have a single instance of each manager in a CA. So, we can make on-demand manager's instances thread-safe. The *command pattern* can be useful when the class making a request should be separated from how that request is executed. We adopt the command pattern in our work to implement a functions for message exchange between the CAs. Fig. 11 shows the code structure of a AbstractCommand class that reduces complexity of decision logic. Each action is implemented in the execute method of its own specific command class. We have implemented 26 classes, which are inherited from AbstractCommand class.

```
public abstract class AbstractCommand {
  protected Vector<ObjectData> cmdRequest;
  // Interface of the abstract command
  public abstract JSONObject encoding();
  public abstract boolean decoding();
  public abstract boolean execute();
  public Vector<ObjectData> getCmdRequest() {
    return cmdRequest;
  }
  public void setCmdRequest(Vector<ObjectData> cmdRequest) {
    this.cmdRequest = cmdRequest;
  }
}
```

Fig. 11. AbstarctCommand class

Fig. 12 shows a partial implementation for *requestAvailableSessionsToJoin* method to process the session join operation. This method encodes a command with parameters as a JSONObject. This JSONObject object is decoded by the decoding method of the concrete command class. When this decoding is completed, the CA will call the execute method.

```
public JSONObject requestAvailableSessionsToJoin
                  (String appPackageID,String fromDevice)
{
  JSONObject body=new JSONObject();
  try {
    body.put("command", "requestAvailableSessionsToJoin");
    body.put("fromdevice", fromDevice);
    body.put("package",appPackageID);
  } catch (JSONException e) {
    e.printStackTrace();
  }
  return body;
}
```

Fig. 12. Representation of command using a JSON object. This figure shows the implementation of *requestAvailableSessionsToJoin* command.

V. Experimental Results

In this section, we evaluate the performance of our system in terms of service discovery for the collaboration services. We conducted a series of experiments for our middleware performance on the multiscreen service platform. In order to test the performance of our system, we used two smart set-tops, which were connected to 150Mbps wireless network connection. Also, the three smart devices (one smartphone and two smart tablets) as mobile-clients were connected to the wireless network as shown in Fig. 13.

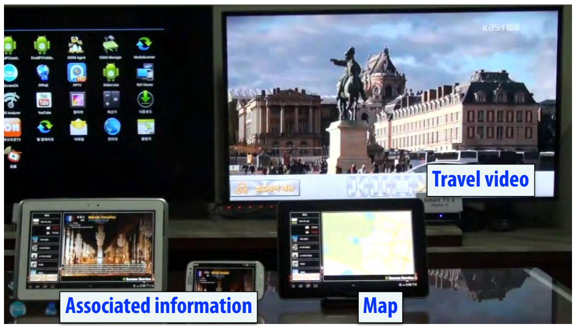

Fig. 13. Collaboration with multiple smart devices: *This figure shows an example of collaboration service among heterogeneous distributed applications.*

In order to evaluate the performance of the middleware, we tested the functions of the n-screen resource discovery. Each smart device includes five different NSAL-based apps. We measured the processing time of service discovery functions every minute during two hours to evaluate its performance. TABLE I shows the test setup for performance evaluation on Android platform.

TABLE I
Test Setup for Performance Evaluation

N-screen device	Processor (RAM)	# of installed apps
Smart set-top 1	Cortex-A9 single core (1GB)	5
Smart set-top 2	Cortex-A9 single core (1GB)	5
Smart tablet 1	1 GHz dual-core (1GB)	5
Smart tablet 2	1 GHz dual-core (1GB)	5
Smartphone	1.4 GHz quad-core (1GB)	5

Fig. 14 presents the processing times according to the n-screen resources. The results show that the proposed middleware provides good performance for the collaboration services. In terms of the performance of n-screen device discovery, it took 0.34 second on average for five devices. And it took 0.36 sec and 0.61 sec on average for 25 apps and 5 collaboration sessions, respectively.

A. Analysis

Our middleware provides good performance for service discovery of n-screen resources, such as, n-screen devices, n-screen apps, and n-screen collaboration sessions. And the proposed middleware maps well to the current smart devices and we have evaluated its performance on four different smart devices such as smart set-top, smartphone and two smart tablets. Furthermore, it is relatively simple to deploy to smart devices on Windows platform as well as Android platform, since it is implemented in Java. This makes it

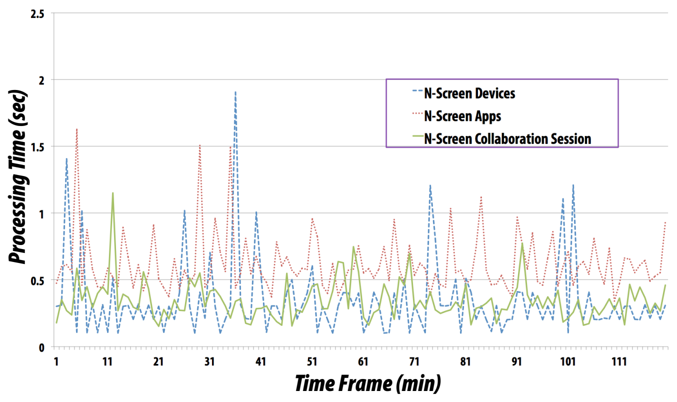

Fig. 14. Performance result of n-screen resource discovery

possible to develop a more flexible multiscreen application for collaboration services.

B. Limitations

Our approach has some limitations. First, the collaboration agent (CA) uses JSON (JavaScript Object Notation) formats for exchanging messages between n-screen devices, thus it is difficult to exchange large messages between the CAs, such as, high-quality photos and videos. We believe that this can be resolved by n-screen application-side implementation. Secondly, the weakness of current system is a its lack of security. However, in terms of collaboration services in multiscreen environments, we should exploit the encryption methods for messages between n-screen apps and CAs, such as, data encryption standard (DES) and advanced encryption standard (AES), for improving the security.

VI. Conclusion

We have presented a multiscreen service platform, which provides collaboration services among associated apps in a smart home. Our system supports *symmetric* collaboration model as well as hierarchical collaboration in peer-to-peer network environment. And it provides seamless service using logical communication among apps and collaboration session management. The proposed middleware can potentially enhance social interactions and user's experiences, extend both social and informational resources available in context, and greatly alter the nature and quality of interactions.

Our middleware greatly improves the service discovery and binding performance, allowing to utilize UDP-based broadcasting and TCP-based messaging. We found that the proposed system provides the scalability of the collaboration services by using the inter-app relationship representations. Moreover, our approach is flexible and maps well to various collaboration services in terms of extension of primitive operations, such as remote execution, collaboration session join/invitation and app migration. In addition, we demonstrate that the proposed system could prove to be flexible in terms of the interoperability among heterogenous apps. So, we believe that our middleware will provide the service scalability with good performance for the multiscreen-based collaboration services.

Acknowledgments

This work was supported by ETRI R&D program ("*Development of Big Data Platform for Dual Mode Batch Query Analysis*, 15ZS1400") funded by the government of South Korea.

References

[1] Samsung multiscreen sdk, http://multiscreen.samsung.com/, 2014.

[2] E. Aarts. Ambient intelligence: A multimedia perspective. *IEEE MultiMedia*, 11(1):12–19, Jan. 2004.

[3] E. Anstead, S. Benford, and R. J. Houghton. Many-screen viewing: Evaluating an olympics companion application. In *Proceedings of the 2014 ACM International Conference on Interactive Experiences for TV and Online Video*, TVX '14, pages 103–110, New York, NY, USA, 2014. ACM.

[4] L. Barkhuus and B. Brown. Unpacking the television: User practices around a changing technology. *ACM Trans. Comput.-Hum. Interact.*, 16(3):15:1–15:22, Sept. 2009.

[5] A. L. M. A. Ehsan Ullah Warriach, Eirini Kaldeli. An interplatform service-oriented middleware for the smart home. *International Journal of Smart Home*, 7(1):115–142, Jan. 2013.

[6] V. Fuentes, N. S. Pi, J. Carbó, and J. M. Molina. Reputation in user profiling for a context-aware multiagent system. In *EUMAS*, 2006.

[7] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 273–284, New York, NY, USA, 2008. ACM.

[8] M. Krug, F. Wiedemann, and M. Gaedke. Smartcomposition: A component-based approach for creating multi-screen mashups. In S. Casteleyn, G. Rossi, and M. Winckler, editors, *Web Engineering*, volume 8541 of *Lecture Notes in Computer Science*, pages 236–253. Springer International Publishing, 2014.

[9] I. Lanese, F. Martins, V. T. Vasconcelos, and A. Ravara. Disciplining orchestration and conversation in service-oriented computing. In *Proceedings of the Fifth IEEE International Conference on Software Engineering and Formal Methods*, SEFM '07, pages 305–314, Washington, DC, USA, 2007. IEEE Computer Society.

[10] A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. In *Proceedings of the 16th European Conference on Programming*, ESOP'07, pages 33–47, Berlin, Heidelberg, 2007. Springer-Verlag.

[11] S. Longo, E. Kovacs, J. Franke, and M. Martin. Enriching shopping experiences with pervasive displays and smart things. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, pages 991–998, New York, NY, USA, 2013. ACM.

[12] A. Lucent. Shaing the future of multiscreen video. *Alcatel Lucent Strategic White Paper*, 2011.

[13] D. Ma, M. Liu, Y. Zhao, and C. Hu. Sscm: Middleware for structure-based service collaboration. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 2224–2225, New York, NY, USA, 2008. ACM.

[14] V. G. Motti and D. Raggett. Quill: A collaborative design assistant for cross platform web application user interfaces. In *Proceedings of the 22nd International Conference on World Wide Web Companion*, WWW '13 Companion, pages 3–6, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[15] A. Nandakumar and J. Murray. Companion apps for long arc tv series: Supporting new viewers in complex storyworlds with tightly synchronized context-sensitive annotations. In *Proceedings of the 2014 ACM International Conference on Interactive Experiences for TV and Online Video*, TVX '14, pages 3–10, New York, NY, USA, 2014. ACM.

[16] H. T. Vieira, L. Caires, and J. a. C. Seco. The conversation calculus: A model of service-oriented computation. In *Proceedings of the Theory and Practice of Software, 17th European Conference on Programming Languages and Systems*, ESOP'08/ETAPS'08, pages 269–283, Berlin, Heidelberg, 2008. Springer-Verlag.

[17] N. Yoshida and V. T. Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *Electron. Notes Theor. Comput. Sci.*, 171(4):73–93, July 2007.