User and Session Mobility in a Plug-and-Play Network Architecture

Mazen Malek Shiaa and Lars Erik Liljeback
Department of Telematics
NTNU University, Trondheim – Norway
malek @item.ntnu.no, liljebac@stud.ntnu.no

Abstract

Network Intelligence is aimed at increasing the awareness and adaptability of today's networks on issues such as service discovery, user behaviour, complexity of applications, ever changing configuration and topology, etc. TAPAS is a network architecture meant for simplifying and automatizing network management, installation, configuration and maintenance. Mobility of users and sessions is important to handle more and more demands regarding user behaviour and complex set of applications. In general, users have one home domain, where they get access to personalised environment and content. They, however, experience different circumstances when login remotely from another domain. Their sessions, on the other hand should always be maintained and registered to provide support for resumption of suspended tasks. User and Session concepts play a crucial role for providing an application platform towards a mobile and demanding end-user.

1 Introduction

TAPAS (**T**elecommunication **A**rchitecture for **P**lug **A**nd Play Systems) is a platform and application development environment based on generic actors in the nodes of the network that can download manuscripts defining roles to be played [1]. The basic model is founded on the concept of the Theatre metaphor, where actors perform according to a predefined manuscript and a director manages their performance. In other words, actors are software components, pieces of code, which represent functionality to be executed at different nodes within the boundary of a network. In order to accomplish different tasks, occasionally complex ones, actors need to interact with each other and change behaviour dynamically. In TAPAS a director is an actor with supervisory status regarding all other actors' plug in and plug out phases, communication, availability of plays, etc. Directors represent also the concept of domain, hence a set of network nodes is managed by a single director. The architecture is covering a set of issues ranging from representation of nodes' capabilities and network resources to QoS awareness and requirements. This architecture is also handling issues of play, set of manuscripts or actor behaviour, management, actor to actor and actor to director communications, and play specification and Plug-and-Play application development.

TAPAS was enhanced and extended to cover, and handle, a range of concepts in order to make it suitable for any kind of network technology and platform. The theme of this paper is about some of these new concepts, namely User and Session. Throughout the paper a logical and intuitive definition of several terms, and message sequence diagrams describing the functionality and management tasks will be provided. Section 2 describes what is meant by Mobility support, while section 3 focuses on the User and Session mobility. Section 4 elaborates all the technical and implementation related issues of mobility support in TAPAS. An application example is presented in section 5, and some test cases are explained in section 6. Conclusion come up in section 7, together with some open questions and future work.

2 Mobility support in TAPAS

As mentioned earlier, actors could be software components running in terminals and network nodes providing the user with certain functionality. Once functionality plugged in, it is possible to configure it, adapt it to the environment, manage it and move it automatically based on certain predefined strategies. This paper focuses on two types of Mobility support, session and user, provided to applications.

Generally speaking, mobility support in the TAPAS is intended to expand to four generic categories: User, Session, Actor and Terminal mobility [2]. User and Session mobility aims at providing the very basic Personal Mobility for users of any system or application based on TAPAS. Users are no longer limited to use certain terminal to access their services provided by their home domain, neither are they required to repeat several tasks in

order to resume some unfinished work from previous sessions. Actor mobility is the support provided for applications to carry out their tasks in a flexible and optimum manner. This could be altered as Software mobility as actors are analogous to software functionality in TAPAS. Terminal mobility, on the other hand assures the continuity of service access while on the move or at location change. This is a complex and rather circumstantial issue that depends heavily on network configuration and terminal capabilities.

To get a wide-ranging view of mobility management in TAPAS an object model and an engineering model should be developed to comprise all needed functionality and support. Figures 1 and 2 depict how mobility is supported in TAPAS. In Figure 1 all objects needed for various mobility kinds are illustrated and related to each other.

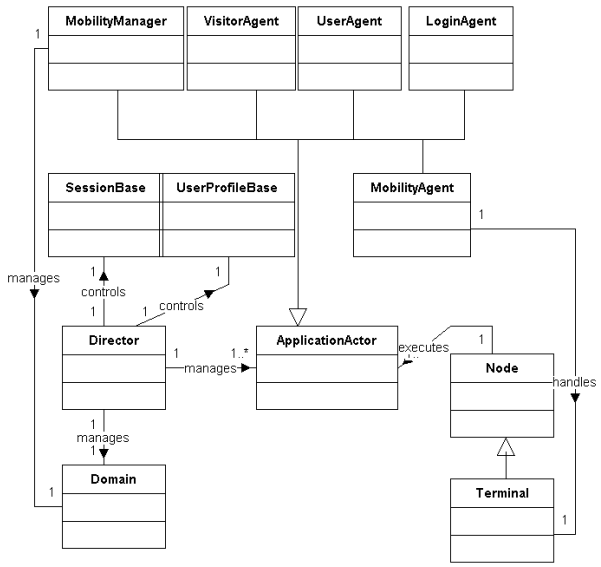

Figure 1. Object model for Mobility support in TAPAS.

Figure 2 gives an engineering model for that. Note that the playing, session and user profile bases are all controlled by the director. Additionally, Mobility Manager and the director could also be co-located in the same node. The devices depicted are characterized by different sets of capabilities, that's why certain application components run at networks nodes instead of user's device. The Mobility Manager is responsible for managing the terminal mobility of wireless devices. In such a configuration any network node could be mobile and being managed by it. For purpose of multi-domain environments the domain's director, as there is only one director per domain, could contact other directors inquiring about visiting user's ID and profile.

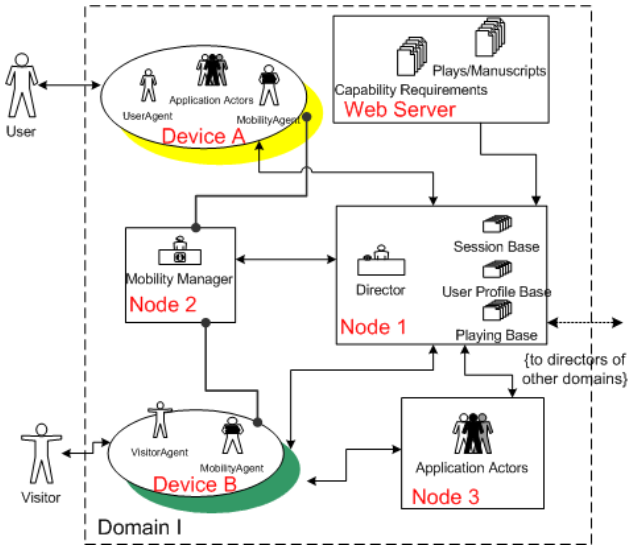

Figure 2. General Engineering model for mobility support in TAPAS.

3 Focus on User and Session Mobility.

What is central to our approach is the terminal identification and mobility of sessions and users. Within this framework terms and notions have been defined and related to active entities in the system. For example:

- User is referred to by an ID and profile,
- user-to-device relation is defined at login phase,
- a user-application interaction is controlled by a *UserAgent*,
- every time a user logs in a user session is maintained,
- personal content is defined by applications and is downloadable,
- applications are dependable on complex capability system regarding device, domain and environment characteristics, etc.

In our approach user's interactions are controlled by *UserAgent* and user sessions are maintained by the director in terms of Session descriptions, which is a detailed sketch of running applications, actors and their related data. All applications will be carried out through the instantiation of actors with certain functionality. Therefore actors need to be maintained in the user's session description and/or profile. Figure 3 demonstrates how user's session is managed by the *UserAgent*, and consequently maintained by the director's data base. An example is provided for a session description and a user profile. In this example application actors are distributed on the

user's device and a network node, a typical example is a chat client and a server, which are managed by the user's agent. When a session is suspended information on every actor's data, e.g. user nick name, online connections with other actors, type of application and additional information for child sessions should be stored. *UserAgent* is completely responsible for the recreation and proper setting of all these application actors.

User's Login phase is central to the definition of user's identity, characterization of device capabilities, resumption of user sessions, and transfer of personal content. It is used by the director to provide users with proper access rights and profiles. Therefore, there are two types of logins in a multi domain environment, as shown in Figure 4. Eventually, users could access their home domain after some inter-director negotiation and authentication. Actors are identified by GAI (Global Actor Identifier) and could move between nodes and running processes. Any specific actor that is required by a user's session has to be moved to the use's new location.

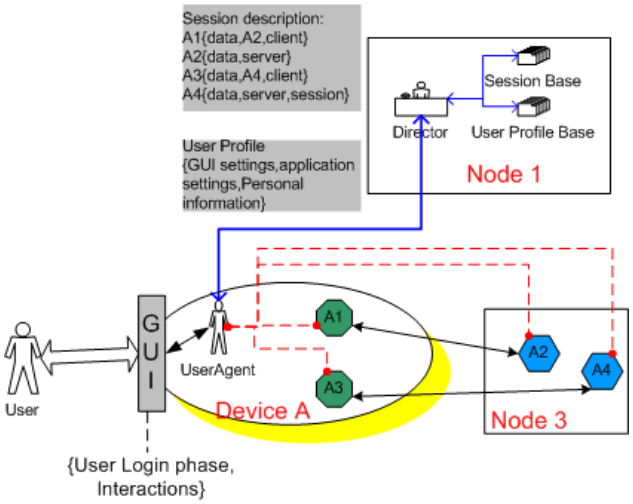

Figure 3. Session mobility: A user is interacting with the system through *UserAgent* and its session is stored in the director's Session Base.

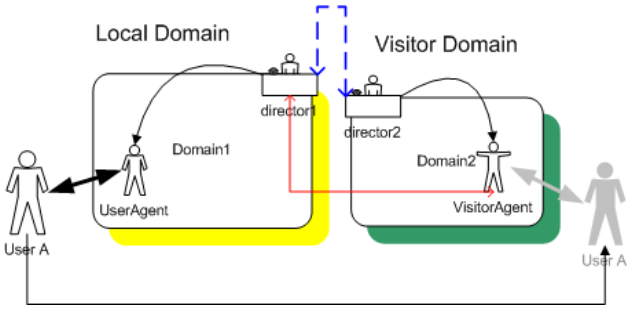

Figure 4. User mobility: User A is accessing his home domain from Domain2.

4 Mobility support functionality

Mobility support is provided to applications, so that it is possible to develop an end-user oriented applications with both user and session mobility enhancements. In these applications users can get access to a personalized environment and could fetch their personal content. So, as users login to the application they can easily interact and perform tasks that could be at any time suspended and resumed later. If applied, user login to his home domain from a visitor domain should also be permitted. In this section we try to handle several implementation related issues of the user and session functionality in TAPAS.

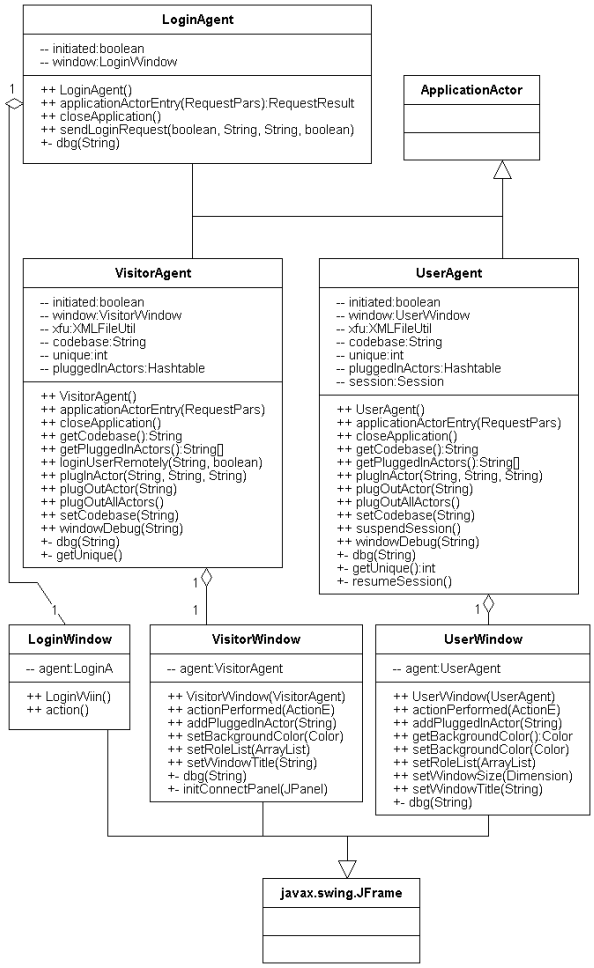

Figure 5. Class diagram for Mobility functionality.

Figures 5 and 6 gives an overview of this functionality and how they are implemented in TAPAS. In Figure 5, *LoginAgent*, *VisitorAgent* and *UserAgent* are derived from the generic *ApplicationActor* object in figure 1. *UserAgent* is responsible for handling the user's interactions with the system while he is at his home domain. *VisitorAgent*, is responsible for a similar task but for visit-

ing users. A single LoginAgent runs at every terminal to handle the login phase of users. All of these classes have their own GUI class; *UserWindow*, *VisitorWindow* and *LoginWindow*, respectively.

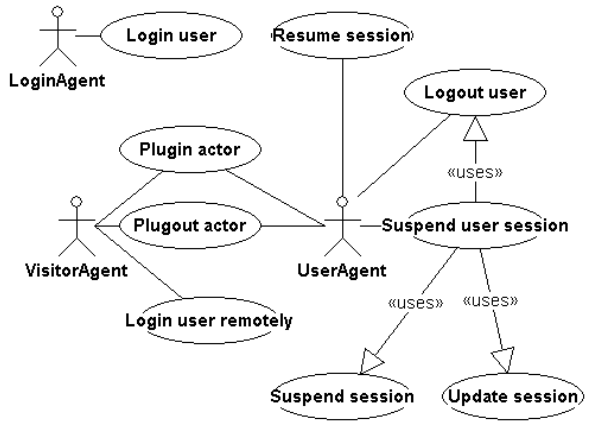

Figure 6. Use cases for mobility functionality.

Figure 6, illustrates how could different use cases handling the behaviour of these newly defined objects be related. These use cases conduct certain types of behaviours that in all define the demanded mobility functionality. Some of these use cases are illustrated in figures 7, 8, 9 and 10.

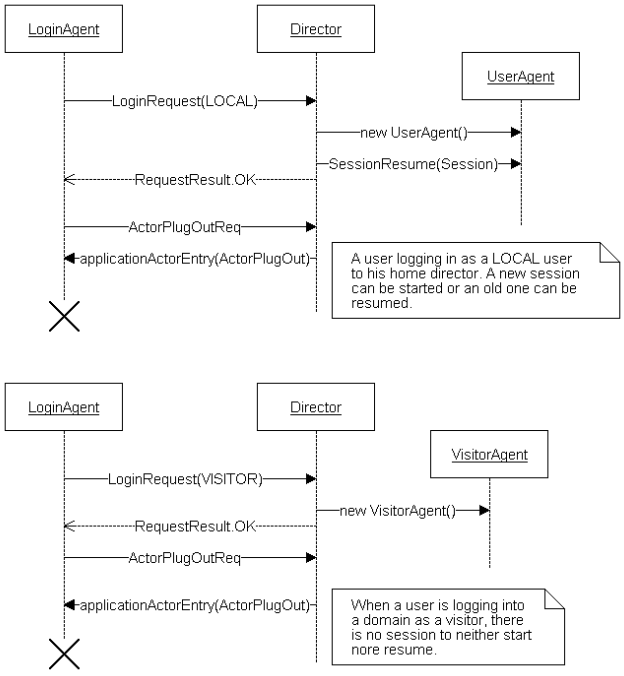

Figure 7. Sequence diagram for logging in users

In these sequence diagrams a logical interpretation of users and their sessions is sketched. Also, the way how TAPAS, the underlying platform, reacts to and handles users' interactions. As a starting point, a user logs in to the system via the *LoginAgent*. As in Figure 7, *LoginAgent* sends a request with a parameter LOCAL or VISITOR, depending on the user's identity. In both cases the Director will decide on granting the required access, for instance it accepts a user as LOCAL and instantiate a *UserAgent* at the node where the user has sent the login request from. The other case would result in instantiating a *VisitorAgent* if the user is not a legitimate user, who's *UserProfile* is not available in the *UserProfile* base.

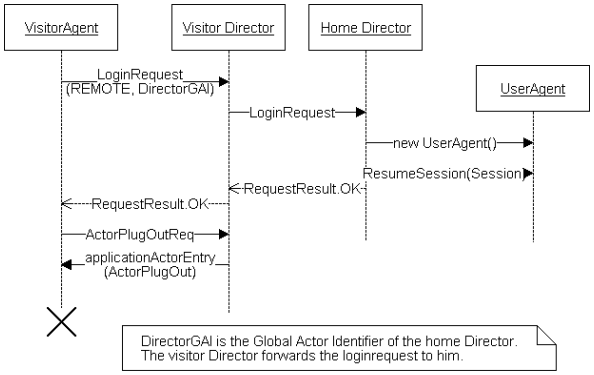

Figure 8. Sequence diagram for remote login.

Another kind of user access is when a visiting user at a visiting domain demands a home domain access, figure 8. In this case *VisitorAgent* issue a login request with two parameters to the director: REMOTE and the home domain's director location. A complete process of inter-director interaction and identity authentication will follow that could result in providing the user with home domain access. As illustrated in the figure, a successful remote login process results in plugging out the *VisitorAgent*. The reason for this is that a *UserAgent* should be instantiated to be capable of handling user's interactions according to a home domain access manner. Different set of applications and access privileges are granted for different user categories, local and visitor.

Session mobility support is achieved through the session suspend and resume procedures. In figure 9 a session suspend starts with a *SuspendSessionReq* request sent from *UserAgent* to all the actors (application actors) that have been plugged in by this *UserAgent*. These actors will reply by sending a description of their own session, which comprises a set of variables needed to reconstruct the session. This specific information on individual actor's session is a dependent part of the application it is representing and how much it is reproducible. For example, a chat client actor would store its server location and user's name. Any suspended session should be registered in the SessionDescripeions base, as mentioned earlier. Session resume, on the other hand, is presented in figure 10. Upon

a new login of a user, who has suspended a session, the director will issue a SessionResumeReq request, which will transfer the session description to the UserAgent. It will manage the whole process of session reconstruction, which starts by making sure that the play required by different application actors are properly available at the directors playing base. The director will receive different ActorPlugIn requests from the UserAgent that are saved as part of the its session description. This process is ended by sending a SessionResumeReq request to these individual actors with required information on their own sessions.

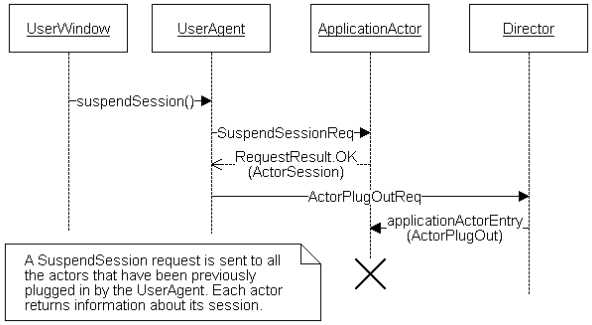

Figure 9. Sequence diagram for Session suspend

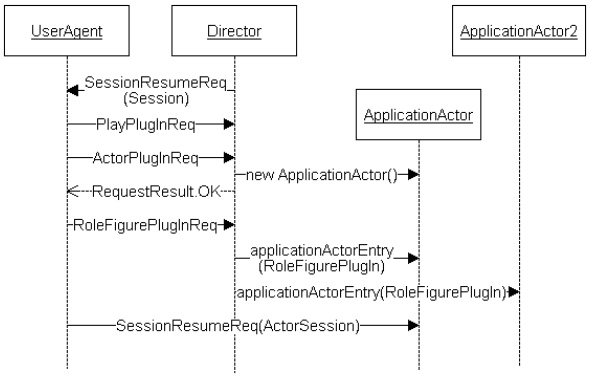

Figure 10. Sequence diagram for Session resume.

```
<UserProfiles>
   <User>
      <Username>user1</Username>
      <Password>password1</Password>
      <BackgroundColor>153 255 153</
BackgroundColor>
      <WindowLocation>326 77</WindowLocation>
      <WindowSize>645 371</WindowSize>
   </User>
   <User>
      <Username>user2</Username>
      <Password>password2</Password>
      <BackgroundColor>0 255 0</BackgroundColor>
      <WindowLocation>100 100</WindowLocation>
      <WindowSize>350 350</WindowSize>
   </User>
</UserProfiles>
```

Figure 11. An example of User Profile.

```
<SessionDescriptions>
  <Session>
    <User>user2</User>
    <ActorInstance>
      <Play>Chat</Play>
      <Version>v1_1</Version>
      <ActorRole>ChatClient</ActorRole>
      <ActorName>ChatClient1</ActorName>
      <RoleSession>
        <Initiator>this</Initiator>
        <Cooperator>Actor://1.1.1.1/pas1/CS
      </RoleSession>
      <ActorState>Username=LarsErik</ActorState>
    </ActorInstance>
  </Session>
```

Figure 12. An example of Session Descriptions.

Figures 11 and 12 gives an overview of the two informational bases: *UserProfile* and *SessionDescriptions*. In our implementation they are stored as two XML files, as the format of the data suggests, and could be very easily extended.

5 The Chat and File Transfer applications

To test the applicability of the mobility functionality support two applications have been developed: Chat and File Transfer applications. These applications comprise two plays each with a set of actor definitions and graphical user interface (GUI).

5.1 The applications

Figure 13 shows how the GUI for the chat application looks like. Users, upon running this application, can connect to available chat servers. Once connected, they are allowed to type instant messages that will be transferred to all users connected to this server. Figure 14 depicts the GUI for the file transfer application. It is simply an application for transferring data files from a node to another node. In both applications a session is maintained by defining one or more actors and store their identity in the session description together with their related information. There is a number of implementation related issues and features that have been discovered and solved out throughout the implementation process. In the chat application, we discovered that a session resumption is not possible if a chat server is allowed to change location, this way a suspended chat client would risk loosing any connection to its server while it is suspended. For this purpose, a limitation on chat servers was set to restrict their mobility. Also, in the file transfer application a file receiver cannot independently resume a suspended file transfer application. It could, however, issue a request to the sender to resume it's own application. Some other cases may require several checks before any session resumption; e.g. check login status of users, check availability of local resources needed for different applications, check servers availability, etc.

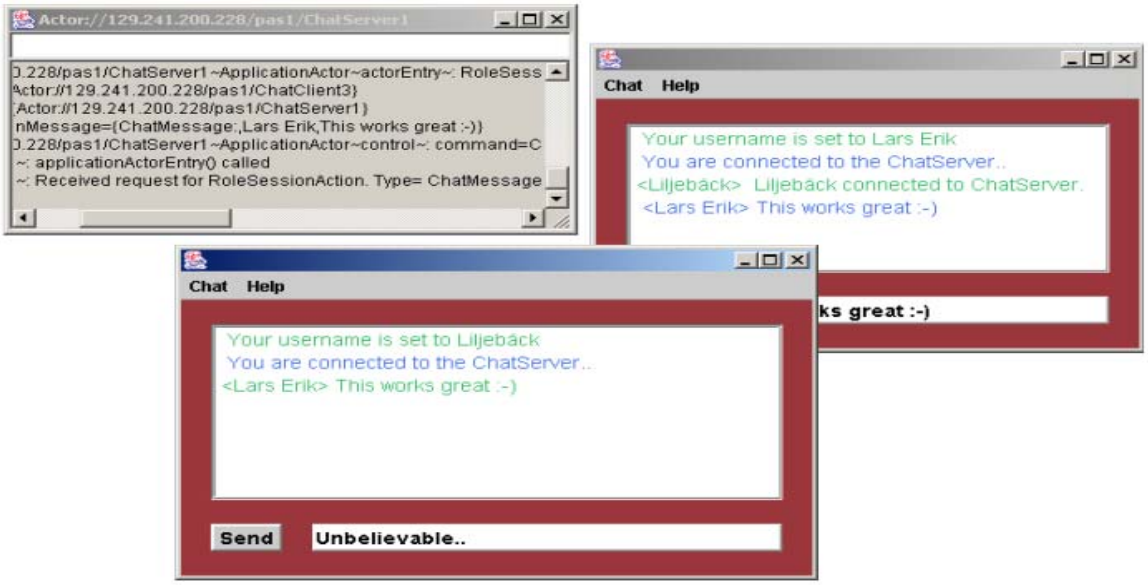

Figure 13. The chat application.

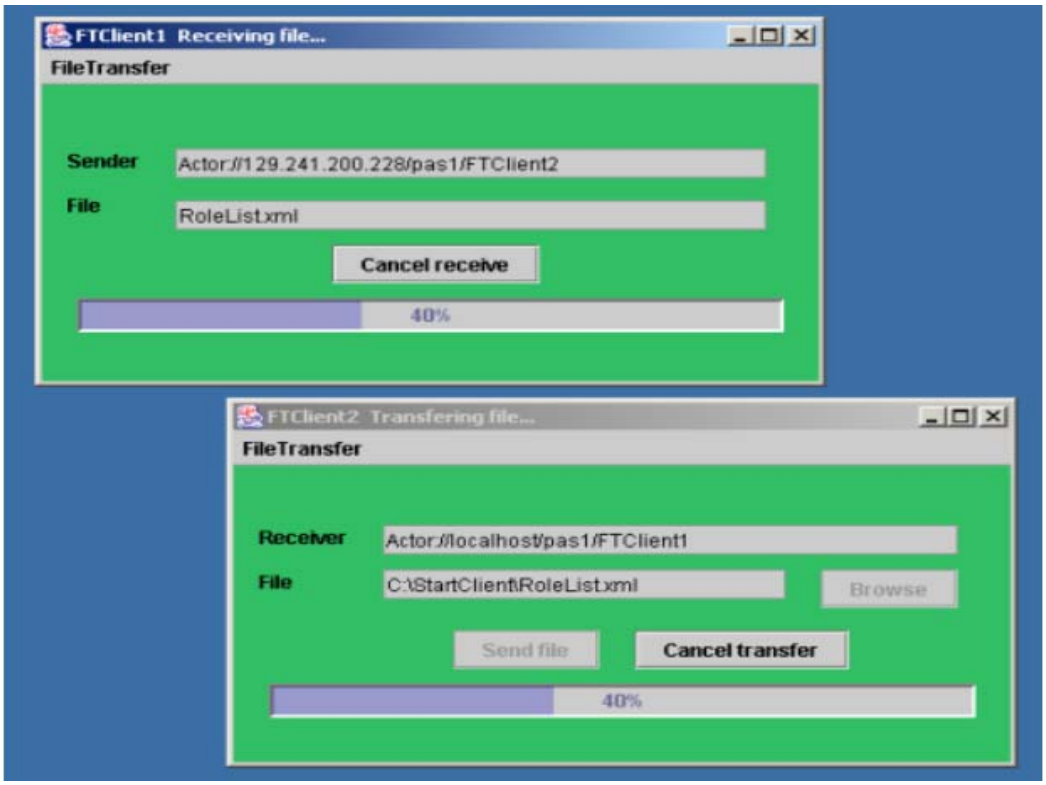

Figure 14. The file transfer application.

5.2 The sessions

Generally, a chat session is suspended by saving the status of the C*hatClient* actor. As in figure 15, upon a request from the *UserAgent* a *ChatClient* disconnects from the *ChatServer* it is connected to and apparently plugs out itself. Before doing so, *ChatClients* saves their state by sending an entry on their active session to be included in the user's S*essionDescriptions*.

When a session that includes a chat application is resumed, *UserAgent* requests the director to plug in a *ChatClient* and tries to pass to it the necessary information to reconstruct its session. SessionResumeReq in this case would include the *ChatServer* location. Figure 16 illustrates this process. Note that some interactions are necessary to accomplish the plug in phase of these actors, but are not related to the Session mobility issue, i.e. they are needed to load a manuscript from an available play in the director's playing base. Every request related to *RoleFigure* and *RoleSession* plug in and setting in figures 15-18 don't contribute to this issue.

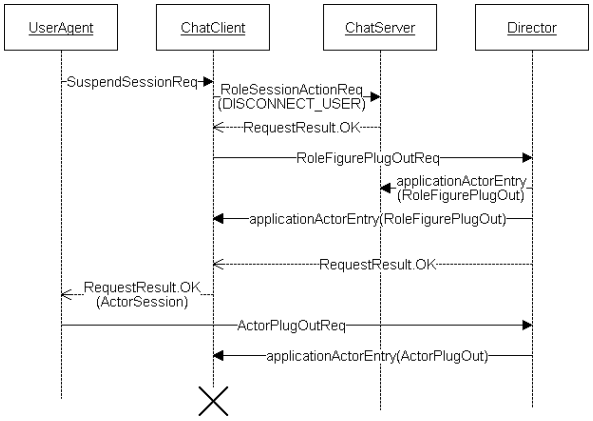

Figure 15. Session suspend for the chat application.

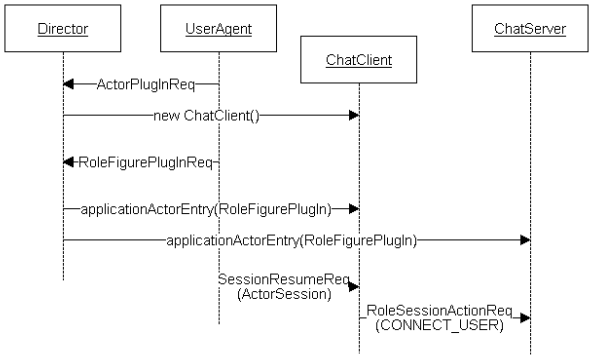

Figure 16. Session resume for the chat application.

Figures 17 and 18 illustrate a similar scenario for the file transfer application. In the first figure, a file transfer application is interrupted by S*uspendSessionReq*, meanwhile a *CancelTransfer* is sent to the other *ChatClient* to stop receiving or sending data.Sequence diagram for the Resume transfer session use case is shown in figure 18. Due to the design of the FileTransfer application a suspended filetransfer can only be automatically resumed if it is the transferring *ftclient* that was suspended. The figure shows what happens if this is the case. A new *ftclient* is plugged in and the rolesession to the receiving *ftclient* is recreated. If the receiving *ftclient* is unavailable the rolesession can not be recreated. The actor's session is resumed when the *UserAgent* sends him a SessionResume request, with the actor's state information added.

At the current implementation stage session resume of file transfer application does not cover all cases, i.e. when resuming a receiving *ftclient* side. There are some features of the underlying platform that make it inefficient to apply such a procedure.

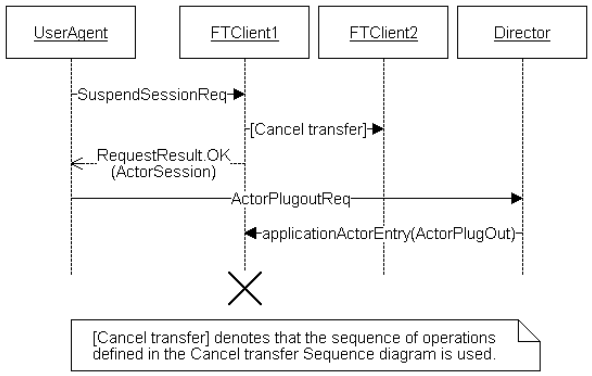

Figure 17. Session suspend for the file transfer application.

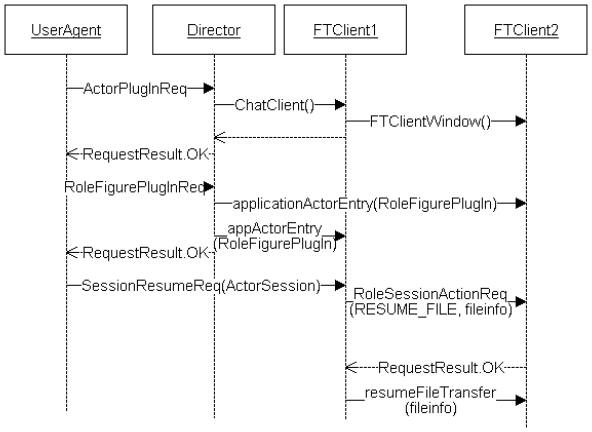

Figure 18. Session resume for the file transfer application.

6 Test Cases

The applicability of the mobility support was tested and several changes have been applied to fit various application requirements. The testing strategy was based on applying different cases where session, users and domains could change or move.

6.1 Testing user mobility

1. *Scenario*: Logon to local *director*. Do not select to resume your session. Change the *UserAgent's* settings and suspend your session. Logon to local director again and this time select to resume your session. *Result*: The settings are stored in your *UserProfile* and set correctly.
2. *Scenario*: Logon to local *director*. Do not select to resume your session. Change the *UserAgent's* settings and suspend your session. Logon to local *director* again on another computer that has the same *director,* and this time select to resume your session. *Result*: The settings are stored in your *UserProfile* and set correctly when you are logged onto the new computer.
3. *Scenario*: Logon to local *director*. Do not select to resume your session. Change the *UserAgent's* settings and suspend your session. Logon as a visitor on a computer with a different *director*. Logon remotely from the *VisitorAgent* to your home domain and this time select to resume your session. *Result*: When you login the settings are stored in your *UserProfile*. The *UserAgent* is plugged in, the user's session is resumed and settings are correct.
4. *Scenario*: Continue where the previous test-case left out. You are logged on to your home domain from another domain. Change the *UserAgent's* settings and suspend your session. Logon to local *director* again on the first computer used and select to resume your session. *Result*: The settings are stored in your *UserProfile* and set correctly when you are logged on.

6.2 Testing session mobility

1. *Scenario*: Logon to local *director*. Do not select to resume your session. Plug-in a *chatclient* actor. Change the username used by the *chatclient* and suspend your session. Logon to local *director* again and this time select to resume your session. *Result*: After the *UserAgent* is plugged in your session is resumed. The *chatclient* is plugged in and the username it set correctly.
2. *Scenario*: Logon to local *director*. Do not select to resume your session. Plug-in a *chatclient* actor. Change the username used by the *chatclient* and suspend your session. Logon to local director again on another computer that has the same director and select to resume your session. *Result*: similar to scenario 2.
3. *Scenario*: Logon to local *director*. Do not select to resume your session. Plug-in a *chatclient* actor. Change the username used by the *chatclient* and suspend your session. Logon as a visitor on a computer with a different *director*. Logon remotely from the *VisitorAgent* to your home domain and select to resume your session. *Result*: After the *UserAgent* is plugged in your session is resumed. The *chatclient* is plugged in (correct username).
4. *Scenario*: Continue where the previous testcase left out. You are logged on to your home domain from another domain. Change the username used by the *chatclient* and suspend your session. Logon to local *director* again on the first computer used and select to resume your session. *Result*: After the *UserAgent* is plugged in your session is resumed. The *chatclient* is plugged in (correct username).

7 Conclusion

In this paper we have presented the approach applied in TAPAS for supporting User and Session mobility. The basic idea was to centre our approach around two informational bases: *UserProfile* and *SessionDescriptions.* The first assigns all information altered as a personal environment and user style related. Information on user's preferred background colours, window sizes, startup applications, different local settings, etc. could be stored in this base. To maintain users sessions, and probably applications sessions, *SessionDescriptions* base hold all the knowledge of actors and their related information while users interact with the system. As such, sessions could be reconstructed and all their suspended actors and tasks could be plugged in again. As a general conclusion we ended up with, mobility and the freedom of session and users is a must in any application focusing on the end users. Also, an object in the model should be assigned the role to maintain and manage this mobility, in our case it was the *director* object.

Improving the interface towards the informational bases could improve the overall functionality. For instance, allowing the *VisitorAgent* to access a list of available domains would simplify the home domain access process. Additionally, applying an actor discovery mechanism could simplify session resume for many applications that are based on client-server communication.

References

1. Aagesen, F. A., Helvik, B. E., uwongse, V., Meling, H., Bræk, R., Johansen, U. *"Towards a plug and play architecture for telecommunications"*. Paper presented at the IFIP TC6 Fifth International Conference on Intelligence in Networks, Bankok, November 1999.
2. Mazen Malek and Finn Arve Aagesen. *"Mobility management in a Plug and Play Architecture"*, IFIP TC6 Seventh International Conference on Intelligence in Networks, Saariselka, Finland, April 2002. Available from: http://www.item.ntnu.no/~plugandplay/publications/