Mobility Support Framework in Adaptable Service Architecture

Mazen Malek Shiaa

Department of Telematics
Norwegian University of Science and Technology (NTNU)
N-7491 Trondheim, Norway

Mazen.Malek.Shiaa@item.ntnu.no

Abstract. Mobility is regarded as *the* most important feature needed to achieve adaptability and flexibility in the executing of service components. As such, service system could be able to cope with the handling of dynamic changes in the availability of resources and position of users. On the other hand, providing user-centric and personal-content driven wide range of services, more commonly wireless ones, to end users regardless of their location and used equipment, seem to be *the* most asked for demand by users. This is the main motivation behind investigating and developing mobility support in any Adaptable Service Architecture. Mobility, in this context, is a feature facilitating the free and coordinated movement of, for instance, users, software components, user terminals, etc. One should always consider the vibrant configuration and settings of not only end-users applications and environment, but also the network resources, components and services. The reason is due to ever changing and increasing demands and requirements in both functionality, security, reliability and QoS. Mobility support in self-managing, dynamically configurable network architecture seems to be even more challenging, however recent development and improvements in network infrastructure show a greater prospect for code-on-demand and adaptive network management. TAPAS, and its mobility handling architecture, presented in this paper, tend to give some answers and take a step towards achieving such goals.

Keywords. Mobility Management, Plug-and-Play, Network Architecture

1. Introduction

There has always been an awareness of the necessity of providing adaptable network services capable of serving users, private as well as business customers, with state-of-the-art information when and where they are needed with a high degree of flexibility, yet simply operated and managed. These services, and the underlying platform or middleware they rely on, have recently been the most important research topic in computer networking. Nowadays, a new network paradigm seems to be a common objective and goal to achieve of many research and development groups—the self-operating, plug-and-play, dynamically configurable network architecture. *TAPAS* (Telematics Architecture for Plug-and-Play Systems) is a research project which aims at developing an *architecture* for network-based service systems with *A)*: flexibility and adaptability, *B):* robustness and survivability, and *C):* QoS awareness and resource control. The goal is to enhance the flexibility, efficiency and simplicity of system installation, deployment, operation, management and maintenance by enabling dynamic configuration of network components and network-based service functionality. So far in this project, a wide range of topics have been dealt with, and many objectives been achieved. Four main architectures have been developed—the basic architecture, mobility handling architecture, dynamic configuration architecture, and the adaptive service configuration architecture— for detailed information see [1,2,3,4,5,6] and the URL: http://www.item.ntnu.no/~plugandplay.

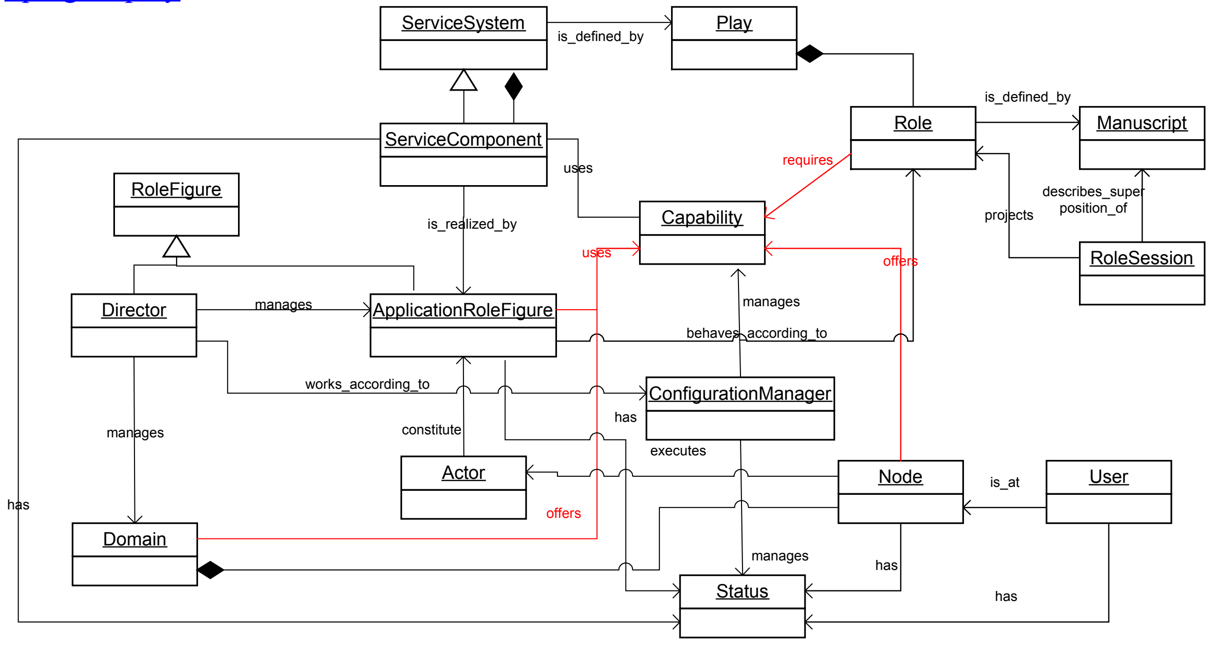


Figure 1. TAPAS basic architecture (Object Model)

TAPAS basic architecture, illustrated in Figure 1, is based on generic *Actors* (software components) in the *Nodes* of the network that can download *Manuscripts* defining *Roles* to be played, each represent different functionality. *Nodes* may be servers, routers and switches, and user terminals, such as telephones, laptops, PCs, PDAs, etc. The model is founded on a theatre metaphor, where *Actors* perform *Roles* according to predefined *Manuscripts*, and a *Director* manages their performance (their plug-in and plug-out phases), and also a *Director* represents a *Domain*. *ServiceSystem* consists of *ServiceComponents*, which are units related to some well-defined functionality defined by a *Play*. A *Play* consists of several *Actors* playing different *Roles*, each possibly having different requirements on *Capabilities* and *Status* of the executing system. A *RoleSession* is a projection of the behaviour of an actor with respect to one of its interacting *Actors*. *Capability* is an inherent property of a *node*. The ability of *Actors* to play *Roles* depends on the defined required *Capability* and the matching offered *Capability* in a *Node* where they intend to execute. *ConfigurationManager* is responsible for obtaining a snapshot of all system resources, and taking decisions on where and how *Capabilities* and *Actors* may be installed and executed. *Capabilities* may be resources (e.g., CPU, hard disk, transmission channels), functions (e.g., printing, encrypting devices), or data (e.g., user login, access rights).

Section 2, will provide an overview of related work regarding network adaptable services and mobility, while Section 0 supplies an overall terminology framework needed to handle mobility as a comprehensive concept. Section 4 gives a view of service management and related considerations in the mobility handling architecture. Section 5 demonstrates the necessity of personal mobility, while section 6 provides a closer look at the software components of the architecture, and how their mobility. Section 7 studies the more commonly addressed concept of terminal mobility, later, Section 8 gives a status of the implementation and some experiences came up throughout the various phases of testing. Finally, Section 9 draws final remarks and conclusion.

2. Related work

There is a wide range of research projects and working groups, both in the academia and industry, working on Adaptable Network Architectures. TINA (Telecommunication Information Networking Architecture) [7] is an effort to put together the best of telecommunications and information technologies aiming at providing solutions to the challenges of developing network information services. It is basically a collection of concepts, tools, and requirement descriptions providing a guideline for such a target. Personal mobility support and other features are being developed and added to the architecture gradually, e.g. [8]. Active Networks are also approaching the same target but from different perspective. They are classified by two approaches: active packets and active nodes. The first builds on the integration and deployment of services in the user flow, while the second is based on deploying services dynamically in nodes. Major research institutes have dealt with this issue with mixed results, see [9,10,11] to check for motivation, results and status in this field, or [12] for a more general and broader survey. Some projects deal with more general issues of flexibility and adaptability in service architecture solutions, e.g. [13], while others focus on the plug-and-play feature applicability in network management functions, e.g. [14]. Many of these solutions apply platforms of either programmable network components, such as [10], while others use mobile agents. A mobile agent is a program, script or package that physically travels around a network, and performs operations on hosts that have agent capabilities. These agents, which operate autonomously, have usually very specific tasks, such as fetching prices of merchandise from on-line stores, or to collect weather information. Apart from interacting with all sorts of operating systems, databases or information systems, agents can also interact with other agents, meeting in agent-gathering places to exchange information. There is a number of different mobile agent architectures, see [15,16]. Although agent technologies have received a lot of attention in recent years, [17] argues that "mobile agency has failed to become a sweeping force of change, and now faces competition in the form of message passing and remote procedure call (RPC) technologies". The very autonomous nature of mobile agents sets them wide apart from TAPAS' request/response interactions, or code-on-demand, although the resulting action might be comparatively equal. Since a TAPAS node can almost be seen upon as a stationary agent, actor mobility is viewed as pair of plug-out and plug-in procedures, in order to move actor instances along their functionality through different nodes where they execute.

Mobile Telecommunication Systems derive and necessitate more elaborated schemes for Personal Mobility. While different systems, such as GSM [8] and in the near future UMTS, provide mobility for users and their terminals in and through enterprise-based domains, service architectures and application platform with high mobility support centred at user and its personal content still lack. The seamless and flexible integration of such application platforms with existing mobile systems is yet a major challenge. A possible solution is the applicability of the Mobile IP concept [19], which is based on two agent processes to take over the routing, the home agent (HA) and the foreign agent (FA). When the mobile host, or user device, leaves its home domain, the HA is informed of this, and the FA of the visited domain relays back to the HA that the host is available in that domain. In our approach of Personal Mobility, the same spirit is maintained, but approached from the service and its provision point of view. User, its session, and subscribed services move along the user and follow its access point, as long as it is possible and allowed in the visited domain. In the future, and in cases of multiple enterprise domains, the approach may be amended to work along the lines of how cellular phones roam networks.

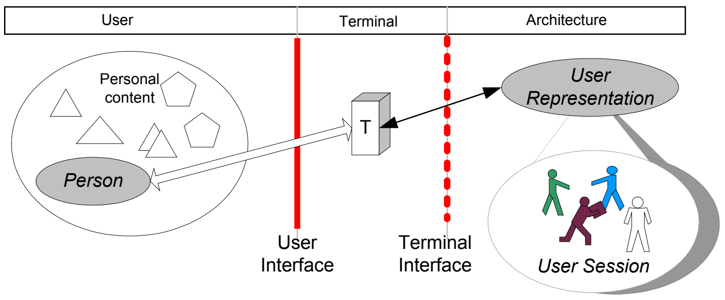


Figure 2. TAPAS Mobility Concept

3. Terminology Framework for Mobility

3.1 Terminology Framework – The Concept

In this section the so-called Terminology Framework for Mobility will be established comprising all the essential definitions needed in providing mobility understanding. TAPAS, in this regard, embraces different mobility features or categories: *Personal (consists of User and User Session mobility)*, *Actor* and *Terminal mobility* [3]. Figure 2 presents a conceptual description of the various terms or entities used to relate these different types of mobility. *User* (*Person*), according to this concept is represented by its *personal contents* and can be related to a *Terminal* (*T*) and be tracked and accessed via a representation of the user (*User Representation*) within the architecture. This double interface approach (*User Interface* and *Terminal Interface*) provides a flexible mechanism to represent users and terminals independently of each other. A user may be represented by a name, while a terminal by a network address. A user may interact with the system, or services, within a defined *User Session*. The movement of user sessions also involves the movement of actors.

3.2 Terminology Framework – The Definitions

Figure 3 shows an extension of the TAPAS basic architecture illustrated in Figure 1, with emphasis on mobility. Below, the newly introduced objects, and other terms specific to the TAPAS terminology will be defined, focusing on the mobility as a concept.

- ***User*** is the end-user of services, which is also called subscriber. In TAPAS there is no distinction between a user of a service, who is carrying out and performing the instantaneous interactions with the service instance, and the subscriber of the service, who owns the subscription contract.
- ***Person*** is a user with personal content.
- ***Personal content*** is the set of user-related data, information and resources that are not part of the service architecture, but might be used or involved in a service interaction.
- ***Node*** is a physical network entity capable of taking part in TAPAS-based services, by running TAPAS support and TAPAS service component(s). This may be directly mapped to PCs, handhelds, mobile phone, or any other device with computing capacity and operating memory capable of running external applications. A node is uniquely specified by its location. ***Terminal*** is one type of node that is associated with end users as their means of accessing services.
- ***Location (Access point)*** is the physical address information. This can be network address, geographical location, etc. A location is used to uniquely address nodes running TAPAS service components.

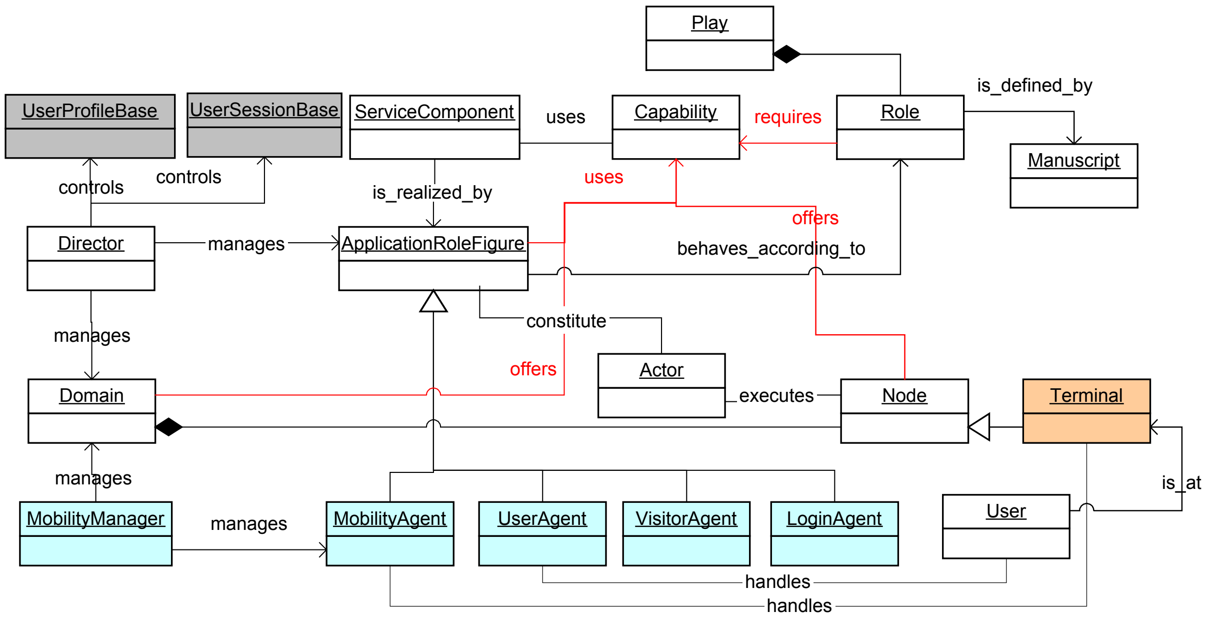


Figure 3. TAPAS Mobility Platform (Object Model)

- ***User Session*** represents the information used by all actor instances involved in the provision of a service for certain user, for certain duration. A user session usually involves the following activities: establishment of the session including the login phase, managing the state of each user's activity, and co-ordination of the network resources used. All user interactions with the system are part of a specific session, however a user may have several simultaneous sessions.
- ***User Session Base*** is the informational or knowledge base responsible for maintaining User Session information.
- ***Domain*** represents a population of actors and/or nodes managed by one director. Domain concept in TAPAS is used to manage and administrate the federation of responsibility between different director objects. In TAPAS two types of domains are distinguished: Home domain and Visitor domain.
- ***Actor*** is the generic object of TAPAS with a generic behavior, which can behave according to a manuscript specifying certain functionality.
- ***Actor Child Session*** represents a session initiated and maintained by an actor, which results in instantiating new actors with their respective data, role-sessions, settings, etc.
- ***Role-Session*** is a projection of the behaviour of an actor with respect to one of its interacting actors. It represents a relationship between two actors.
- ***User Profile*** includes the user information relevant to the provision of services (such as user location, subscribed services, permissions, constraints, etc.). It also includes the set of optional preferences and setting attributes of services associated with the user.
- ***User Profile Base*** is the informational or knowledge base responsible for maintaining User Profile information.
- ***Actor Mobility*** stands for the movement of instantiated functionality at a node that is executed by an actor. This implies a change and update to the actor location-specific information.
- ***Role-Session Mobility*** stands for the re-instantiation of role-sessions of moved actors by re-creating them at the new location where the moved actor is re-instantiated.
- ***Terminal Mobility*** is the movement of terminals and change of their location while maintaining access to services and applications.

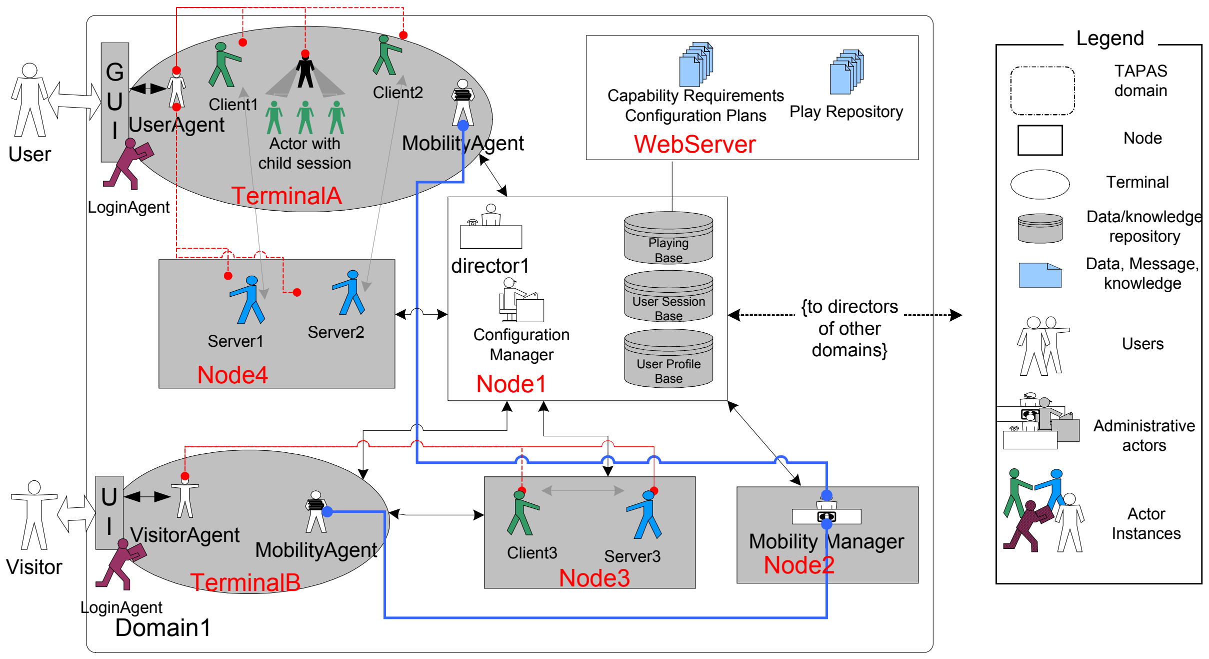


Figure 4. TAPAS Mobility Architecture (Engineering Model)

- ***Personal Mobility*** is the utilization of services that are personalized with end user's preferences and identities independently of both physical location and specific equipment.
- ***User Session Mobility*** is the re-instantiation or resumption of Applications, Actors and Role sessions enabling a user to carry on with suspended user sessions.
- ***User Mobility*** is the seamless access of subscribed services at different access points.
- ***Login Agent*** is the component of the architecture that supports the entering of a user to the service(s) environment under managed permissions, constraints and optional preferences, which are described in the user profile.
- ***User Agent*** is the component of the architecture that is responsible for managing user interactions with its home domain. Home domain is a domain where there exists a user profile for the user.
- ***Visitor Agent*** is the component of the architecture that is responsible for managing user interactions with its visitor domain. Visitor domain is a domain where there doesn't exist a user profile for the user.
- ***Mobility Agent*** is the component of the architecture that is responsible for managing terminal's location-related information. It performs location updates when a terminal changes its location.
- ***Mobility Manager*** is responsible for managing actors and terminals connectivity and mobility.

3.3 Terminology Framework – The Engineering model

In order to obtain a clearer view of how these various terms and concepts are related, an engineering viewpoint should be established. Figure 4 presents an engineering model of TAPAS mobility platform illustrated in an example domain, which associates architecture components and objects with the defined terms and roles. It shows a single domain managed by a director, and includes four nodes, two user terminals, and a web server. Instances of services are executing distributely on terminals and nodes by different actor instances, i.e. clients, servers, and actors with child sessions.

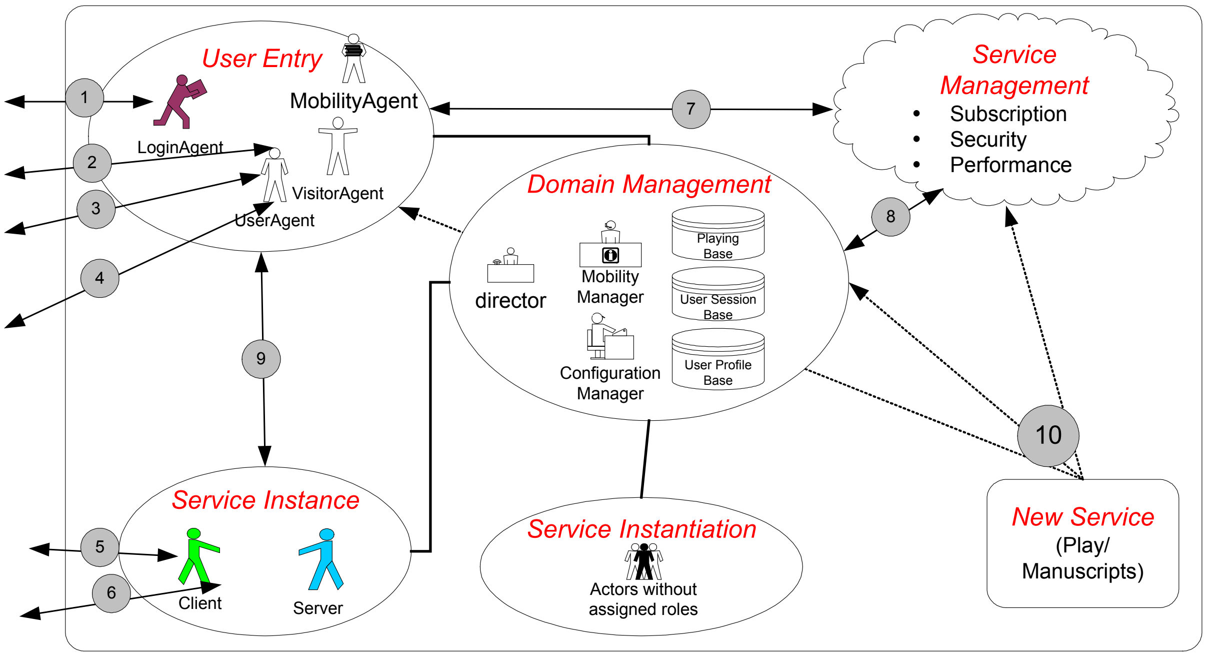


Figure 5. Management Considerations of Service Provision

4. Mobility Support and Service Management

Any network architecture that provides services to end-users should establish a clear view of how these services be subscribed to by users, utilised by operators, and developed by service developers. The topic of service management has been handled and experienced by various enterprise business models, that's why it is out of the scope of this paper. However, providing mobility support or mobility feature to a service architecture affects the way services may be managed or maintained, and to what extent their applicability may evolve. Based on the main concept of TINA service architecture, mobility support changes into the service management of TAPAS is illustrated in Figure 5. Basically, the picture is divided into six main parts or components; *User entry* (executes at the user's terminal), *Domain management* (includes all supervisory components, such as director, configuration manager, and knowledge bases), *Service instantiation* (where instantiated functionality resides), *Service instance* (the part of the service handling user's interactions and instructions), *Service management* (which corresponds to the service provider domain), and *New service* (the introduction of new play and its corresponding functionality). The various phases shown as numbered arrows may be summerised in the following: (1) User login or entry to the system is accomplished by *LoginAgent* component, (2) Service requests by the user are managed through its *UserAgent* or *VisitorAgent*, (3) Administration by the user, e.g. changing settings and preferences, is achieved through *UserAgent* or *VisitorAgent*, and saved in *UserProfile,* (4) *UserSession* management is controlled by *UserAgent* or *VisitorAgent*, (5) Service management by the user may be managed through the service actors executing certain role-figures, e.g. clients, (6) User interactions with the service instance are performed through the service actors, (7) Administration by the service provider results in changes in the user entry components, (8) Updating user subscription information should take place in the *UserProfile*, (9) Interactions between user entry and service instance components, e.g. when the user changes some settings and preferences, and at last (10) Providing new service implies a change in both user entry, service management and domain management components. These phases give a guideline to achieve a mobility support inline with a general service management platform, allowing for the integration with any enterprise model.

5. Personal Mobility

Personal mobility as defined earlier comprises two types of mobility: User and User Session mobility. Mobility in this context is a support provided to applications and services, 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 applicable, user login to *home domain* from a *visitor domain* should also be permitted. Generally speaking, personal mobility is based on the following assumptions: 1) *User* is referred to by *Name* and *User Profile,* which is active through a *User Interface* (at a *Terminal* through *Terminal Interface*), 2) User-to-terminal relation is defined at *login phase*, 3) Director maintains *User Profiles* in *UserProfileBase,* which contains information on user settings, preferences and personal data, 4) *UserAgent* or *VisitorAgent* controls the user interactions with the system, and maintain *User Session* for each *login phase*, and 5) Director and Configuration Manager decide how and where different service components (application role-figures) be instantiated depending on play/configuration requirements and terminal/node available capabilities. However, *UserAgent* should keep track of all actor instances that belong to a *User Session*. Director maintains *UserSession* related information in *UserSessionBase.* The following subsections demonstrate user session and user mobility using general working examples.

5.1 User Session Mobility

User session mobility is aimed at providing users with greater flexibility in terms of suspending and resuming their execution of services. Figure 6 illustrates how user session is managed by *UserAgent*, and consequently maintained by the director's *UserSessionBase*. This figure is also a general example of a probable service execution scenario in TAPAS. At first, user *UserA* login to the system at *TerminalA* through its *Graphical User Interface* (*GUI*) and executing *LoginAgent.* As a consequence of this login phase, the user granted a home domain type of access to its subscribed services—*Service1* and *Service2* defined by *Play1* and *Play2*, respectively. *UserAgent* is instantiated and *UserA Session* is maintained. The connectors between the actors *Client1* and *Client2* in *TerminalA* and the actors *Server1* and *Server2* in *Node4*, indicate that they are inter-related by role-sessions. *Server1, Client1* and *Client2* are instantiated application role-figures *Role11*, *Role12*, and *Role22* from *Play1*, while *ActorA, A1, A2* and *A3* instantiated role-figures *RoleA1, RoleA11, RoleA12* and *RoleA13* from *Play2*, all respectively. The dotted connectors between *UserAgent* and *Server1, Client1, Client2* and *ActorA* indicate that they belong to this user session. Note that *Server2* is not maintained by this session. An example is provided for *UserSession* description. Typical example of such applications is a multiple room chat application as *Service1* (*Sever2* is being owned by another user), and Internet browser application as *Service2*. User session is updated regularly via the *update_session* request, while suspended via *suspend_session* request. *UserAgent* sends these requests to *director1* containing information on every instantiated actor data, e.g. user name, role-sessions, type of application and information about actor child sessions. Actually, *UserAgent* initiates a suspend session procedure upon a request from the user. Consequently, it requests every instantiated actor for its settings and child session information. Secondly, when a user wants to resume a suspended session—assuming *UserA* has moved to another location, for instance *TerminalA'*—director will inform the newly instantiated *UserAgent* about any suspended sessions by sending a *resume_session* request. *UserAgent* will then try to re-instantiate all actors, and their corresponding sessions, as indicated in that request. The type and configuration of applications, availability of service definitions, and changed set of available capabilities will determine the way and extent of resumed session.

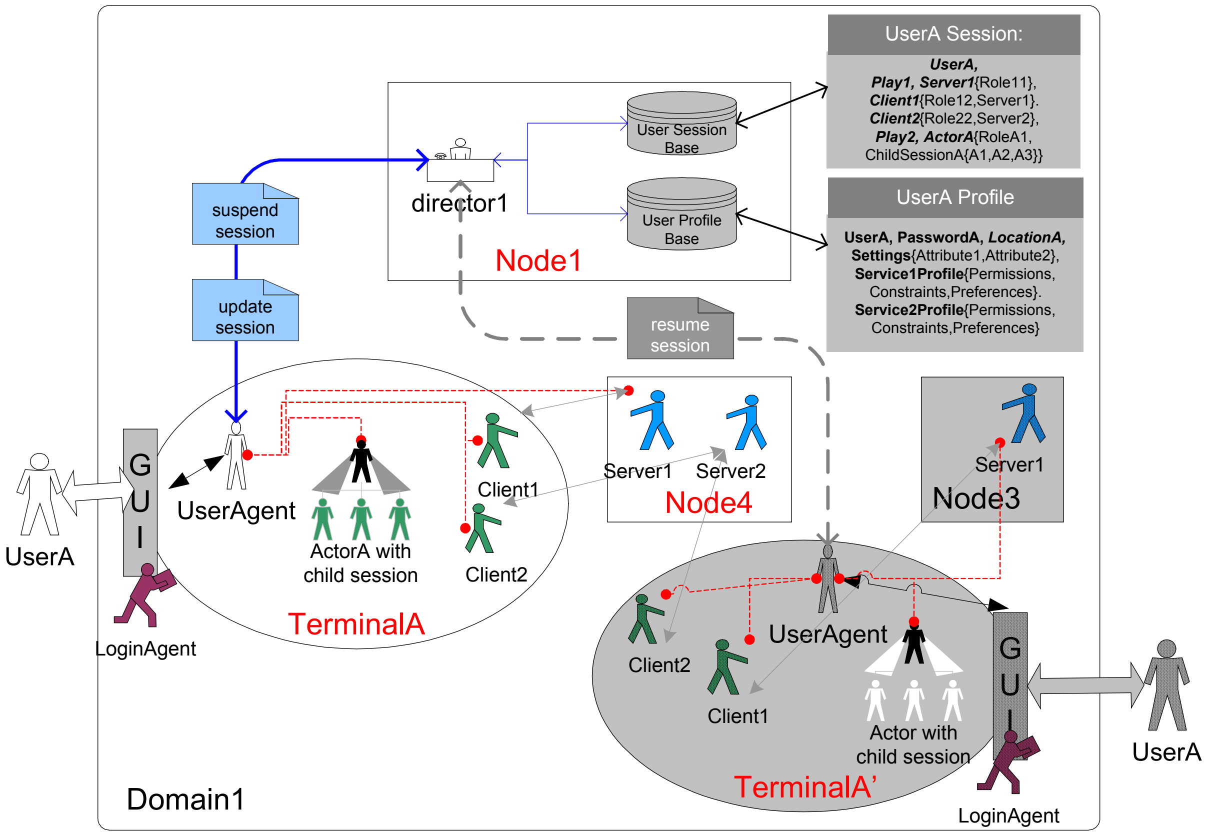


Figure 6. UserSession Mobility in TAPAS

5.2 User Mobility

User Mobility is aimed at providing users with greater flexibility in terms of roaming among different domains while maintaining seamless access to their subscribed services. Figure 7 illustrates how user mobility is achievable in TAPAS. The numbered steps determine a logical order of actions to understand the operation of this feature. Assume first, (1) *UserA* accessing his home domain, *Domain1*. As described previously, (2) *UserAgent* will contact the director of the domain, *director1*, to get all user profile and relevant personal content information from *UserProfileBase*. The requests *request_profile* and *update_profile* ask for and change user's profile, respectively. This step might involve the resumption of suspended sessions as indicated in the previous section. In (3), the user will move and try to access the same set of services from a visitor domain, *Domain2*. Upon login phase, (4) this user is assigned a *VisitorAgent*, as there is no user profile for this user, and hence he granted only visitor status services and applications. When the user tries to access his home domain subscription (services and personal content), (5) *VisitorAgent* indicate this to the director, *Director2*. After a director-to-director negotiation and a sort of authentication process in (6), there arises a possibility to grant this user a home domain type of access in (7). As indicated by this logic, the login phase determines the access type a specific user is given, and therefore whether a *UserAgent* or *VisitorAgent* is instantiated. Further organization and regulation of services, access rights, permissions, allowed operations and activities may all be configured as a domain-based relation and vary among different business models

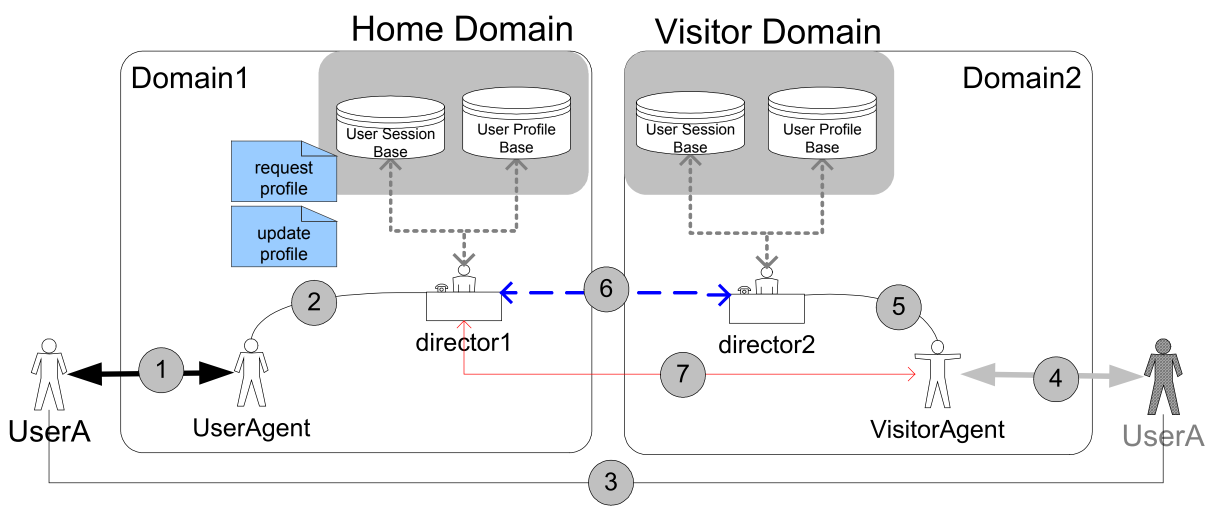
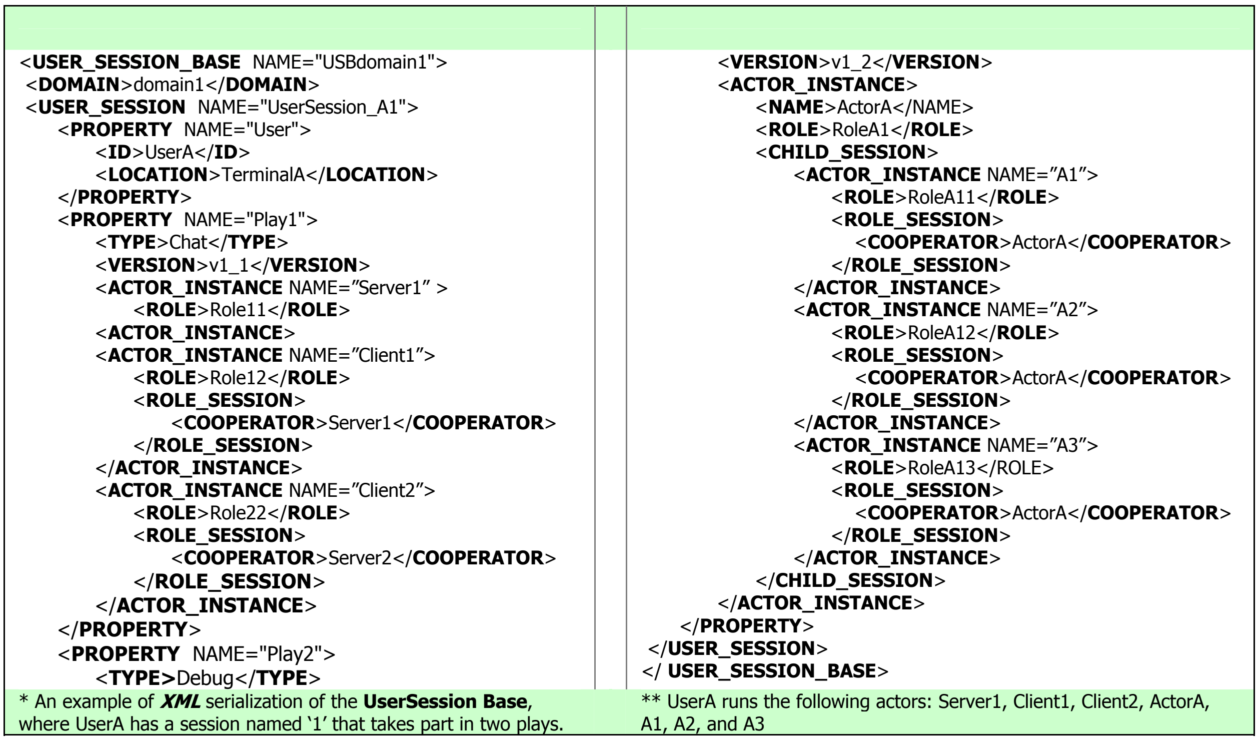


Figure 7. User Mobility in TAPAS

5.3 Knowledge bases (*UserSessionBase* and *UserProfileBase*)

For automated, modular, and extendable solution for such knowledge bases, an XML-based databases and requests are proposed. Figure 8 and Figure 9 give an overview of the two knowledge bases: *UserSessionBase* and *UserProfileBase*, respectively. In the implementation platform they are stored as two XML files, and could be easily extended. Based on previous subsections, an example is worked out for a probable service execution scenario in *domain1*.



```
<USER_SESSION_BASE  NAME="USBdomain1">
 <DOMAIN>domain1</DOMAIN>
 <USER_SESSION  NAME="UserSession_A1">
     <PROPERTY  NAME="User">
         <ID>UserA</ID>
         <LOCATION>TerminalA</LOCATION>
     </PROPERTY>
     <PROPERTY  NAME="Play1">
         <TYPE>Chat</TYPE>
         <VERSION>v1_1</VERSION>
         <ACTOR_INSTANCE NAME=”Server1” >
             <ROLE>Role11</ROLE>
         <ACTOR_INSTANCE>
         <ACTOR_INSTANCE NAME=”Client1”>
             <ROLE>Role12</ROLE>
             <ROLE_SESSION>
                 <COOPERATOR>Server1</COOPERATOR>
             </ROLE_SESSION>
         </ACTOR_INSTANCE>
         <ACTOR_INSTANCE NAME=”Client2”>
             <ROLE>Role22</ROLE>
             <ROLE_SESSION>
                 <COOPERATOR>Server2</COOPERATOR>
             </ROLE_SESSION>
         </ACTOR_INSTANCE>
     </PROPERTY>
     <PROPERTY  NAME="Play2">
         <TYPE>Debug</TYPE>
         <VERSION>v1_2</VERSION>
         <ACTOR_INSTANCE>
             <NAME>ActorA</NAME>
             <ROLE>RoleA1</ROLE>
             <CHILD_SESSION>
                 <ACTOR_INSTANCE NAME=”A1”>
                     <ROLE>RoleA11</ROLE>
                     <ROLE_SESSION>
                       <COOPERATOR>ActorA</COOPERATOR>
                     </ROLE_SESSION>
                 </ACTOR_INSTANCE>
                 <ACTOR_INSTANCE NAME=”A2”>
                     <ROLE>RoleA12</ROLE>
                     <ROLE_SESSION>
                       <COOPERATOR>ActorA</COOPERATOR>
                     </ROLE_SESSION>
                 </ACTOR_INSTANCE>
                 <ACTOR_INSTANCE NAME=”A3”>
                     <ROLE>RoleA13</ROLE>
                     <ROLE_SESSION>
                       <COOPERATOR>ActorA</COOPERATOR>
                     </ROLE_SESSION>
                 </ACTOR_INSTANCE>
             </CHILD_SESSION>
         </ACTOR_INSTANCE>
     </PROPERTY>
 </USER_SESSION>
</ USER_SESSION_BASE>
```

* An example of ***XML*** serialization of the **UserSession Base**, where UserA has a session named ‘1’ that takes part in two plays.

** UserA runs the following actors: Server1, Client1, Client2, ActorA, A1, A2, and A3

Figure 8. General XML serialization of *UserSessionBase*.

Generally, these databases are self-explained and may be viewed as a step-by-step approach to (re)construction of user sessions and specification of user profile information. All the requests involved in the user session and user mobility procedures (*update_session*, *suspend_session, resume_session, update_profile,* and *request_profile*), may be extracted from these databases by matching the corresponding tags or fields in the XML file.

```
<USER_PROFILE_BASE NAME="UPBdomain1">
 <DOMAIN>domain1</DOMAIN>
 <USER NAME="UserA">
    <PROPERTY NAME="Password">
        <VALUE>****</VALUE>
        <VALID>010104</VALID>
    </PROPERTY>
    <PROPERTY NAME="Location">
        <VALUE>local</VALUE>
    </PROPERTY>
    <PROPERTY NAME="Settings">
        <ATTRIBUTE NAME="BGColor">
            <VALUE>White</VALUE>
        </ATTRIBUTE>
        <ATTRIBUTE NAME="WSize">
            <VALUE>Large</VALUE>
        </ATTRIBUTE>
    </PROPERTY>
    <PROPERTY NAME="Service1">
        <PERMISSIONS>
            <VALUE>Owner</VALUE>
```

*** An example of ***XML*** serialization of the **UserProfile Base**, which includes the UserProfile for a single user, UserA.

```
        </PERMISSIONS>
        <CONSTRAINTS>
            <VALUE>LocalAccess</VALUE>
        </CONSTRAINTS>
        <PREFERENCES>
            <VALUE>Empty</VALUE>
        </ PREFERENCES >
    </PROPERTY>
    <PROPERTY NAME="Service2">
        <PERMISSIONS>
            <VALUE>Temp</VALUE>
        </PERMISSIONS>
        <CONSTRAINTS>
            <VALUE>LocalAccess</VALUE>
        </CONSTRAINTS>
        <PREFERENCES>
            <VALUE>Empty</VALUE>
        </ PREFERENCES >
    </PROPERTY>
 </USER>
</ USER_PROFILE_BASE>
```

*** UserA has a subscription to Service1 (specified by Play1), and Service2 (specified by Play2).

Figure 9. General XML serialization of *UserProfileBase*

6. Actor Mobility

Actor mobility stands for the movement of instantiated functionality at a node along its properties, such as behaviour, capabilities, role-sessions, etc., in a transparent manner for all other actors. Actors need to move due to several reasons, e.g. changed capability requirements, deterioration in resource availability, dynamic change in configuration, change in functionality, or implications of terminal mobility. Moved actors need to be able to carry on their functionality after being re-instantiated at a different location.
Before handling this type of mobility, and reasoning about the move procedure itself, an actor model should be constructed. Therefore, an actor, as described by TAPAS basic architecture is defined by the following parts: *1)* set of interfaces or role-sessions, *2)* behaviour definition that has state, *3)* set of capabilities, *4)* queue of incoming requests, and *5)* set of methods accessible by other actors at specific interfaces. Mobility in this context will be achieved by re-instantiation of actors with these parts. However, there might be different strategies of how to perform it, for instance the queue of already queued requests may be carried out, moved, or simply dismissed upon a move procedure. Principally, move procedure is defined by, or equivalent to, a sequence of *ActorPlugOut*, *ActorPlugIn*, *CapabilityChange*, *CreateInterface*, and *BehaviourChange* procedures, which are part of the basic architecture, and used to destroy an actor, instantiate it, update its capabilities, set a role-session with another actor, and change the manuscript it executes, respectively. *ActorMove* request, which triggers an actor move, supplies the new location where the actor should be plugged in, while its interface, behaviour, capability, queue, and method definition should be preserved from its previous instance. To allow for different interpretations by run environments, programming languages, and operating aspects, certain conditions must be specified that will control this procedure. A basic set of conditions may be: A) Capability and Interface parts may be reconstructed through applying supplementary *CapabilityChange* and *CreateInterface* procedures, B) Behaviour part must be the same, and state information may be transferred using *Behaviour-Change*, and C) queue and method parts will be dismissed.

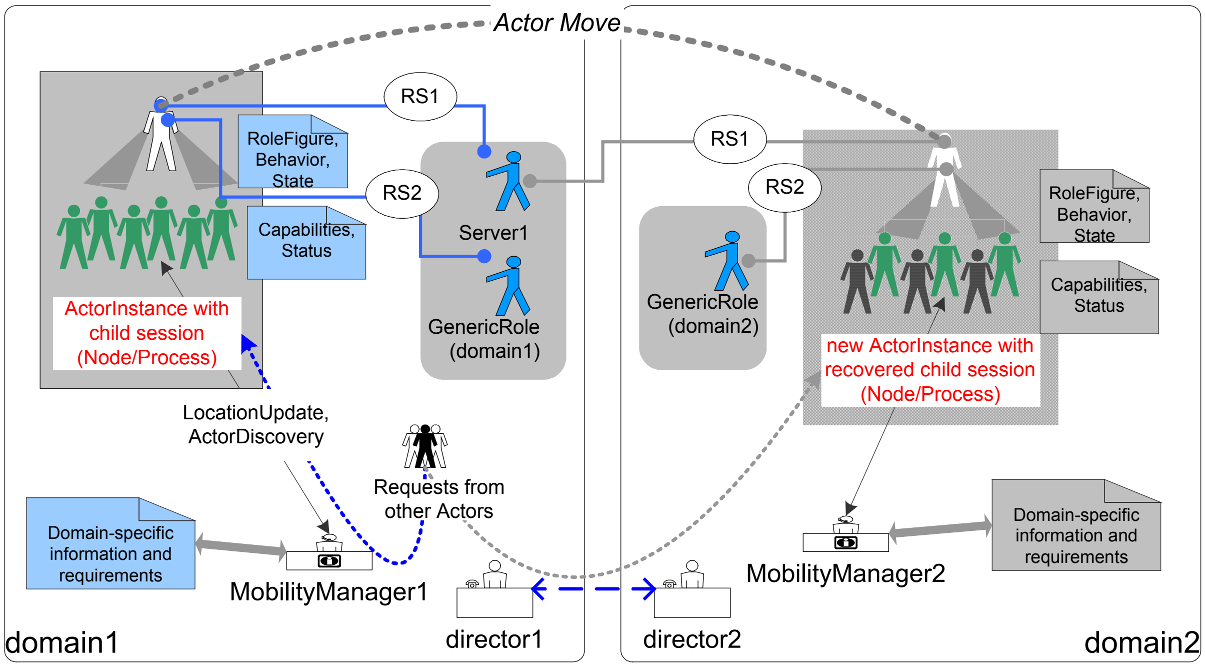

Actor Move
RS1
RS2
RoleFigure, Behavior, State
Capabilities, Status
Server1
GenericRole (domain1)
ActorInstance with child session (Node/Process)
LocationUpdate, ActorDiscovery
Requests from other Actors
Domain-specific information and requirements
MobilityManager1
director1
domain1
RS1
RS2
GenericRole (domain2)
RoleFigure, Behavior, State
Capabilities, Status
new ActorInstance with recovered child session (Node/Process)
Domain-specific information and requirements
MobilityManager2
director2
domain2

Figure 10. Actor Mobility in TAPAS

Figure 10 presents a general scenario for actor mobility that involves two different domains, based on conditions A, B and C. A general actor instance, with possible child session consisting of several actor instances, is moved across two domains, *domain1* and *domain2*. The actor has two role sessions, *RS1*and *RS2*, with *Server1* and another actor instance, referred to as *GenericRole* to indicate its limited availability in *domain1* as a special purpose actor, e.g. providing directory and domain name server. As explained earlier, an actor model consists of several parts; behaviour, role-sessions, capabilities, etc. When moving actors these parts must be recovered, however certain ones might not be recoverable, as for instance certain capabilities may not be available at the new location, or specific role-sessions are no longer relevant. In Figure 10, *RS2* at the new location has been recovered to point to another *GenericRole* actor instance, the one available in *domain2*. Also, the moved actor child session couldn't be fully recovered; certain actor instances have been assigned different functionality and/or capabilities illustrated by different colours. When an actor moves from one location to another, which is characterized by node address and process id, *MobilitManager* is responsible for managing the accessibility to this actor. The actor is required to inform the *MobilityManager* about this movement by initiating a *LocationUpdate* procedure. Meanwhile, requests from other nodes addressed to this actor should be preceded by an *ActorDiscovery* procedure, which is executed through the corresponding *MobilityManager* in a domain. The actor movement within a domain is complying with a set of domain specific settings and requirements. Upon entering another domain, an actor may be accessed through a director-to-director authentication process, as it has been passed to the responsibility of another *MobilityManager*.
Figure 11 illustrates a possible message sequence of an actor changing and updating its location, while some node performs an *ActorDiscovery* procedure looking up for this actor. Upon receiving the move request, *ActorMove*, a series of actions need to be performed to re-instantiate the actor instance at the new location. First, an *ActorPlugIn* procedure is carried out, then recovering of capabilities, role-sessions, behaviour should follow via *CapabilityChange, CreateInterface,* and *BehaviourChange* procedures, respectively. State in the *BehaviourChange* request refers to the state of the actor at which the move procedure have been activated. Second, *MobilityManager* is updated via *LocationUpdate* procedure.

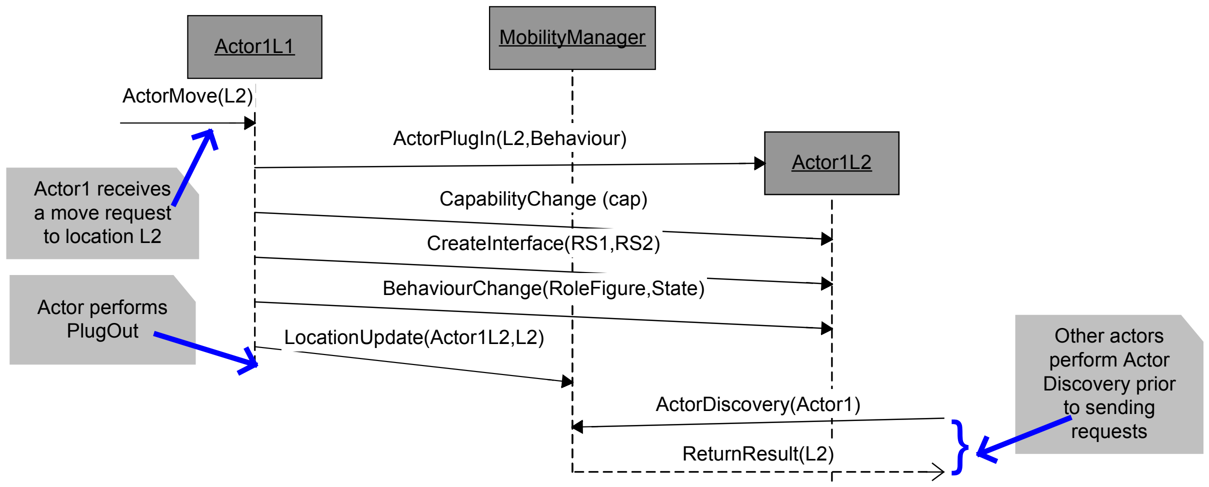

Actor1L1
MobilityManager
Actor1L2
ActorMove(L2)
ActorPlugIn(L2,Behaviour)
CapabilityChange (cap)
CreateInterface(RS1,RS2)
BehaviourChange(RoleFigure,State)
LocationUpdate(Actor1L2,L2)
ActorDiscovery(Actor1)
ReturnResult(L2)
Actor1 receives a move request to location L2
Actor performs PlugOut
Other actors perform Actor Discovery prior to sending requests

Figure 11. Message sequence of a general actor move: "assume ***Actor1*** (***Behaviour, cap,*** {***RS1,RS2***}) at ***L1*** moves to ***L2"***

7. Terminal Mobility

In TAPAS, terminals realize the interface towards the end user, whilst nodes are viewed fixed as seen from their location point of view, though they might be given changeable or dynamic network addresses. The mobility as a feature is mainly provided for terminals, as end users want to access their subscribed services while on the move at different locations. This has been clearly seen in the engineering model of TAPAS, where at each node the support runs at a distinct network address or location, and terminals execute a *MobilityAgent* responsible for tracking their location. To achieve mobility management for these moving terminals we need to keep track of their movements. So a manager should be responsible for updating the locations of all nodes that participate in a possible TAPAS service. This central and supervisory agent will be referred to as *MobilityManager*, and runs at an address known to all other nodes, for instance its network location may be part of a configuration file. *MobilityAgent* will issue *LocationUpdate* procedure, upon changing terminals location, and *NodeDiscovery* procedures, once a communication is required with other terminals or nodes. Figure 12 demonstrates a general case of terminal mobility. A terminal moves from one domain to another, from *doamin1* to *domain2*, while its *MobilityAgent* ensures that *MobilitManager* is updated on this movement. However, when it reaches the limit of one domain, or the so-called *out-of-coverage*, it considered as inaccessible. Meanwhile, requests from other nodes addressed to this terminal should be preceded by a *NodeDiscovery* procedure, which is executed through the corresponding *MobilityManager* in a domain. Upon entering another domain, and similar to the scenario studied in User Mobility, a terminal may be allowed to access certain services based on a director-to-director authentication process. *MobilityManagers* operate according to a set of domain specific set of settings and requirements, which govern the privileges and access rights specific user or terminal may have.

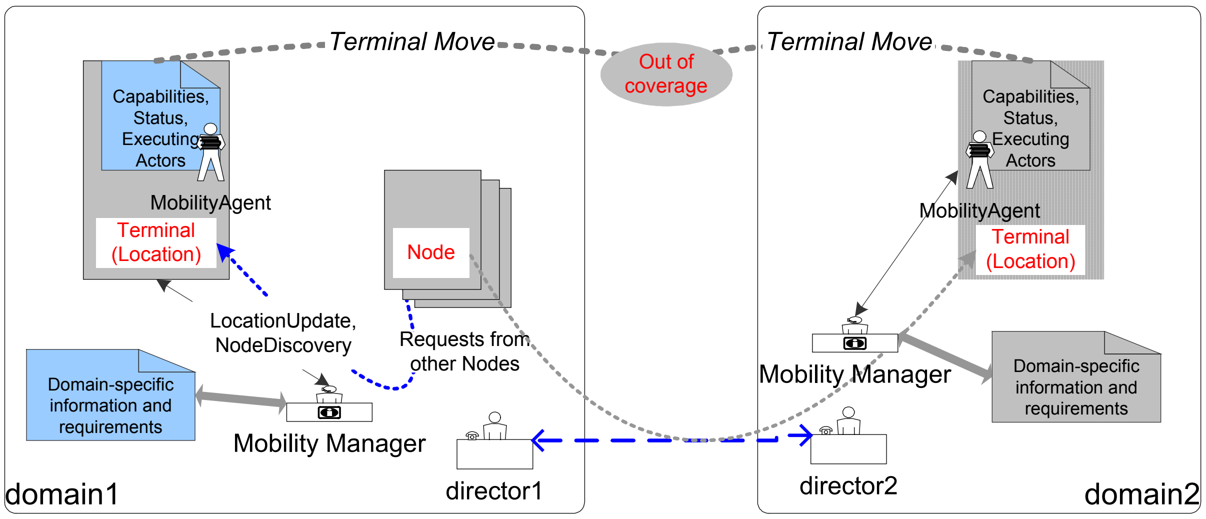

Terminal Move
Out of coverage
Terminal Move
Capabilities, Status, Executing Actors
MobilityAgent
Terminal (Location)
Node
LocationUpdate, NodeDiscovery
Requests from other Nodes
Domain-specific information and requirements
Mobility Manager
director1
domain1
Capabilities, Status, Executing Actors
MobilityAgent
Terminal (Location)
Mobility Manager
Domain-specific information and requirements
director2
domain2

Figure 12. Terminal Mobility in TAPAS

Figure 13 illustrates a possible message sequence of a terminal updating its location, while some node performs a *NodeDiscovery* procedure looking up for this terminal.

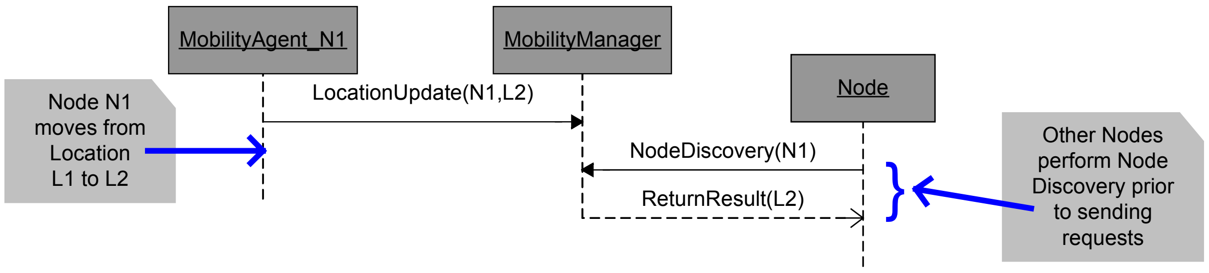

MobilityAgent_N1
MobilityManager
Node
Node N1 moves from Location L1 to L2
LocationUpdate(N1,L2)
NodeDiscovery(N1)
ReturnResult(L2)
Other Nodes perform Node Discovery prior to sending requests

Figure 13. Message sequence of a general Terminal move

8. Implementation issues and Experiences

The TAPAS architecture requires a support system for software development, deployment, execution and management. Parts of this support functionality have been implemented using JAVA RMI and Web technologies as a means for service definition, update and discovery. Some of the mobility features have been implemented, while others undergo redefinition and partial implementation. *User* and *user session mobility* have been implemented and demonstrated in both fixed and wireless environments [4,6]. 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. Several test cases have been constructed and tested.
Terminal mobility has been so far limited to the introduction of two kinds of objects: *MobilityManager* and *MobilityAgent*, in order to track and control terminals and their location change. The TAPAS platform support has been extended to give support for limited capability, small user devices, such as PDAs using J2ME and based on sockets as a communication model. Dynamic connection of wireless terminals, along side static connections of nodes, is achieved using specific network routines, for instance checking the status of a network socket or pinging a host. Wireless terminals are characterized by their movement and varying quality of connectivity, which occasionally may take them into out-of-coverage areas. Therefore any loss of connection must be tracked, and if possible recovered, and consequences must be taken into consideration. Such consequences might be marking unconnected actors and delet-

ing their role-sessions. Also, for practical reasons, separate configuration information on the operating wireless environment and user behaviour must be maintained to allow for efficient setting of such routines. For instance, in a highly dynamic and wireless environment it is advantageous to check connection status more frequently than in more static and less mobile conditions. There is already a solution for a domain of PDAs with tiny downloadable applications operating by means of wireless LAN connections. Basically the mobility schemes for Actor and Terminal, studied in the previous sections, seem to be well fitted and properly integrated. However, more vibrant situations and different environment settings need to be tested for better exploitation of resources.
The present actor realisation does only give a simplified *Actor mobility*, which is a simplified pair of plug-out and plug-in procedures. Although this seems to be adequate for wide range of services, new and more powerful actor model is needed to experience the full power of Actor Mobility. A new Actor model is being developed and formalised to cope with such need. Additionally, there are certain issues need to be studied to improve the overall Actor mobility. For example, the so-called Actor Proxy may be developed to simplify the actor discovery procedures, so that an actor sends all requests to this proxy to try the last known addressed actors before initiating that procedure upon failure of delivery. On the other hand, Actor Replicas may be instantiated for certain type of actors, e.g. all actors running on wireless terminals. These replicas will try to recover the behaviour of actors, which loose connectivity.

9. Conclusion

In this paper the TAPAS Mobility Handling Architecture has been thoroughly presented. First, Mobility as a comprehensive concept was dealt with, and categorized into: Personal, Actor, and Terminal mobility types. Terminology Framework has been established to give a precise definition for all elements of the targeted architecture. This was a crucial phase of addressing the mobility issue, as it tends to be quite general and wide-ranging topic. The paper, as promised at the beginning, succeeded in providing several answers regarding the issue of mobility handling and its support in Adaptable Service Architecture, in particular, but not necessarily limited to, the TAPAS terminology. The mentioned types of mobility were carefully studied and looked at, and easy-to-interpret examples were worked out to give a general understanding for the proposed solutions. As a final remark, mobility support is a best-effort type of support, may be provided with varying degree of fulfilment to certain entities of the network architecture. As have been demonstrated in all types of mobility, *UserSession* may or may net be fully recovered, *User* and its *Personal content* may or may not be supported at another domain, *Actor* and all of its parts may or may not be entirely re-instantiated, and finally, *Terminal* may be denied of services because it is considered inaccessible. The paper also established a basic comparison between these handling mechanisms and other applied techniques in certain research areas.

References

1. Aagesen, F. A., Helvik, B.E., Wuvongse, V., Meling, H., Bræk, R. and Johansen, U., “Towards a Plug and Play Architecture for Telecommunications”, *Proc. 5th IFIP Conf. Intelligence in Networks* (*SmartNet’99*), Bangkok, Thailand, Kluwer Academic Publisher, November 1999
2. Aagesen, F. A., Helvik, B. E., Anutariya, C., and Shiaa M. M., “On Adaptable Networking” *The 2003 International Conference on Information and Communication Technologies (ICT 2003),* April 8-10, 2003
3. Shiaa M.M and Aagesen. F.A. “Mobility management in a Plug and Play Architecture”, *Proc. IFIP 7th Int’l Conf. Intelligence in Networks* (*SmartNet’2002*), Saariselka, Finland, April 2002. Kluwer Academic Publishers

4. Shiaa M.M and Aagesen. F.A. "Architectural Considerations for Personal Mobility in the Wireless Internet", *Proc. IFIP TC/6 Personal Wireless Communications (PWC'2002)*, Singapore, October 2002. Kluwer Academic Publishers
5. Aagesen, F. A., Anutariya, C., Shiaa, M. M. and Helvik, B. E., "Support Specification and Selection in TAPAS", *Proc. IFIP WG6.7 Workshop on Adaptable Networks and Teleservices*, September 2002, Trondheim, Norway
6. Shiaa M.M. and Liljeback L.E., "User and Session Mobility in a Plug-and-Play Network Architecture", *Proc. IFIP WG6.7 Workshop on Adaptable Networks and Teleservices*
7. Berndt H., Darmois E., Dupuy F., Inoue Y., Lapierre M., Minerva R., Minetti R., Mossotto C., Mulder H., Natarajan N., Sevcik M., and Yates M., "The TINA Book", *Prentice Hall Europe* 1999.
8. Tzifa, Louta, Liossis, Kaltabani, Polydorou, Demestichas, and Anagnostou, "Open Service Architecture with Personal Mobility Support and Accounting and Charging Capabilities", *European Multimedia, Embedded Systems and Electronic Commerce Conference EMMSEC99*, Stockholm, Sweden, 21-23 June 1999.
9. Massachusetts Institute of Technology, Active Networks, http://www.sds.lcs.mit.edu/activeware/ [Accessed May 2003]
10. Colombia University, Department of Computer Science, NetScript, http://www.cs.columbia.edu/dcc/netscript/ [Accessed May 2003]
11. U.S. Department of Defence, Advanced Technology Office, http://www.darpa.mil/ato/programs/activenetworks/actnet.htm [Accessed March 2003].
12. Tennenhouse D.L., Smith J.M., Sincoskie D., Wetherall D.J and Minden G.J., "A Survey of Active Network Research", *IEEE Communications*, Vol. 35, No 1, 1997.
13. The IBM autonomic computing project http://www.research.ibm.com/autonomic/ [Accessed May 2003]
14. Bieszczad A., Pagurek B. and White T., "Mobile Agents for Network Management", *IEEE Communications Surveys*, Vol. 1, No. 1, 1998.
15. University of Maryland, Baltimore County, Lab for Advanced Information Technology, KQML Web, http://www.cs.umbc.edu/kqml/ [Accessed May 2003]
16. Stanford University, Department of Computer Science, Knowledge Sharing Effort, http://www-ksl.stanford.edu/knowledge-sharing/ [Accessed May 2003]
17. Reilly, David, "Mobile Agents - Process migration and its implications", http://www.davidreilly.com/topics/software_agents/mobile_agents/ [Accessed May 2003]
18. Mouly M. and Pautet M., "The GSM System for Mobile Communications", *Mouly & Pautet*, 49, rue Louise Bruneau, F-91120 PALAISEAU – FRANCW, 1992.
19. G. Coulouris, J. Dollimore, T. Kindberg, "Distributed systems, concepts and design", *third edition, Addison-Wesley*, 2001