MOBILITY SUPPORT FRAMEWORK IN ADAPTABLE SERVICE ARCHITECTURE

Mazen Malek Shiaa
Department of Telematics, Norwegian University of Science and Technology
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 execution of service components. As such, service systems 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 important objective of such a feature. 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 the ever changing and increasing demands and requirements in 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.

Key words: Mobility Management, Plug-and-Play, and 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

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35703-4_21 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.

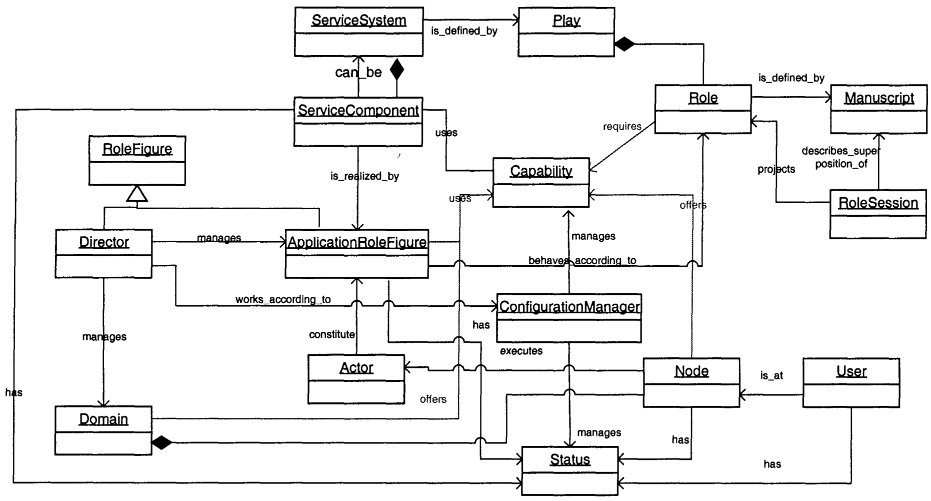

Figure 1. TAPAS basic Architecture (the Object Model)

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. The model is founded on a theatre metaphor, where *Actors* perform *Roles* according to predefined *Manuscripts*, and a *Director* (one in a *Domain*) manages their performance. *ServiceSystem* consists of *ServiceComponents*, which are units related to some well-defined functionality defined by a *Play*. 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.

Section 2, will provide an overview of related work, while Section 3 supplies an overall terminology framework needed to handle mobility. Section 4 gives a view of service management and related considerations. Section 5 demonstrates the necessity of personal mobility, while section 6 provides a closer look at the software components of the architecture, and examines their mobility. Section 7 studies the concept of terminal mobility, later, Section 8 gives a status of the implementation and some experiences came up throughout the various phases of testing.

2. RELATED WORK

There is a wide range of research projects and working groups 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. 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. 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. 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”.

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 centered at the 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). In our approach of Personal Mobility, the same spirit is maintained, but approached from the service 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.

3. TERMINOLOGY FRAMEWORK FOR MOBILITY

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].

3.1 Terminology Framework – The Concept

Figure 2 presents a conceptual description of the various terms or entities used to relate the 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.

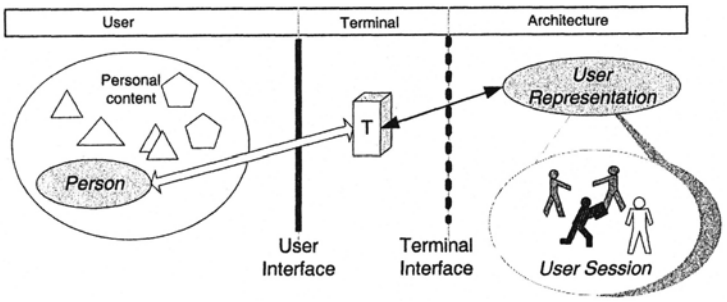

Figure 2. TAPAS Mobility Concept

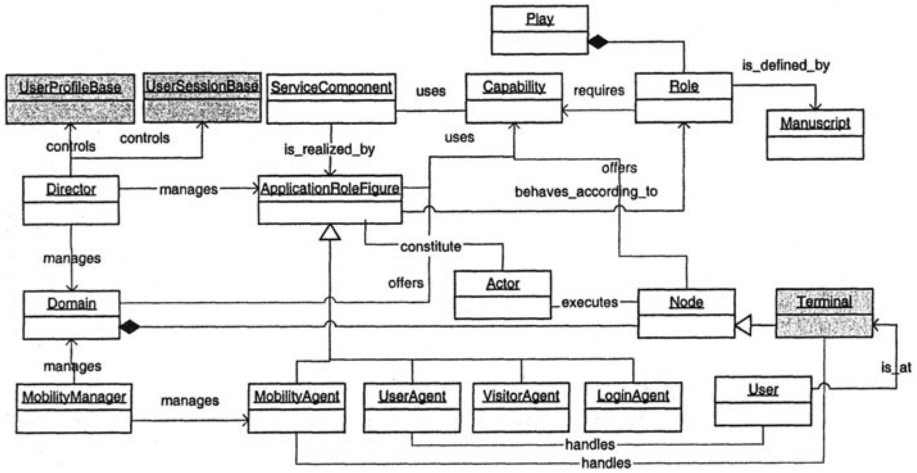

Figure 3. The Object Model of the TAPAS Mobility Platform

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.

- ***User*** is the end-user of services, who is also the service subscriber.
- ***Person*** is a user with some personal content.
- ***Personal content*** is the set of user-related data, information and resources that might be used by the service architecture.
- ***Node*** is a physical network entity capable of running TAPAS-based services, and uniquely specified by a location. ***Terminal*** is a type of node that is associated with the end users as their means of accessing services.
- ***Location (Access point)*** is the physical address information. This can be network address, geographical location, etc.
- ***User Session*** represents the information used by all actor instances involved in the provision of a service for certain user, for certain duration. 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.
- ***Actor*** is the generic object of TAPAS with a generic behavior.
- ***Actor Child Session*** represents a session initiated and maintained by an actor, which results in instantiating new actors.
- ***User Profile*** includes the user information relevant to the provision of services (such as user location, subscribed services, permissions, constraints, etc.).
- ***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 by updating 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.
- ***Terminal Mobility*** is the movement of terminals and change of their location while maintaining access to services and applications.
- ***Personal Mobility*** is the utilization of services personalized with end user's preferences and identities independently of 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 sessions.
- ***User Mobility*** is the seamless access of services at different access points.

- ***Login Agent*** supports the entering of a user to the service(s) environment under managed permissions, constraints and optional preferences.
- ***User Agent*** 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 responsible for managing user interactions with its visitor domain. Visitor domain is a domain where there is no user profile for the user.
- ***Mobility Agent*** 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 mobility.

4. MOBILITY 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, utilized by operators, and developed by service developers. The topic of service management has been handled and experienced by various enterprise business models. On the other hand, providing mobility support or mobility feature to service architecture affects the way services may be managed or maintained. Based on the main concept of TINA service architecture, the service management of TAPAS is illustrated in Figure 4. Basically, the picture is divided into six main parts or components: *User entry*, *Domain management*, *Service instantiation*, *Service instance*, *Service management*, and *New service*. The various phases shown as numbered arrows may be summarized 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.

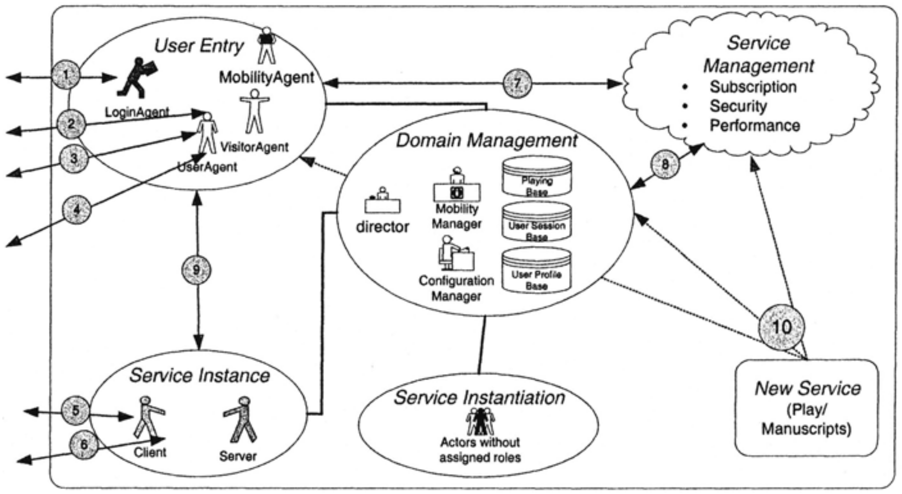

Figure 4. Management Considerations of Service Provision

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. 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.* [4] dealt with this type of mobility in detail, however certain enhancements to the approach have emerged, which with a general description are provided in the following subsections.

User session mobility is aimed at providing users with greater flexibility in terms of suspending and resuming their execution of services. Figure 5 illustrates how user session is managed by *UserAgent*, and consequently maintained by the director's *UserSessionBase*. This figure includes a login phase (*UserA* at *TerminalA*) and a moving phase (from *TerminalA* to *TerminalA'*). The figure shows an example of subscribed services: *Service1* and *Service2* defined by *Play1* and *Play2*. Relations between actors are indicated by connectors, e.g. *Server1* and *Client1*, while dotted connectors between *UserAgent* and some actors indicate that they belong to one user session, e.g. *Server2* is not maintained by this session. An example is provided for *UserSession* description. 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. *resume_session* request is used to resume the sessions. 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, e.g. *ActorA*'s session may not fully recovered at the new terminal. Examples of XML serialization of the User's session and profile are sketched later in Table 1 and 2.

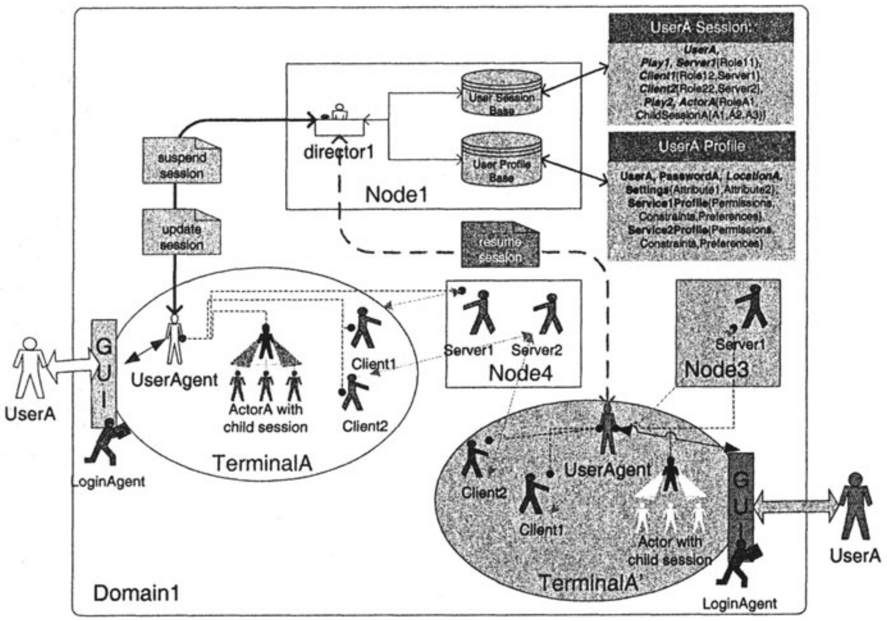

Figure 5. UserSession Mobility in TAPAS

Table 1. General MXL serialization of UserSessionBase

```
<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>
      <COOP>Server1</COOP>
     </ROLE_SESSION>
    </ACTOR_INSTANCE>
    <ACTOR_INSTANCE NAME="Client2">
     <ROLE>Role22</ROLE>
     <ROLE_SESSION>
      <COOP>Server2</COOP>
     </ROLE_SESSION>
    </ACTOR_INSTANCE>
   </PROPERTY>
    <PROPERTY NAME="Play2">
     <TYPE>Debug</TYPE>
     <VERSION>v1_2</VERSION>
     <ACTOR_INSTANCE NAME="ActorA">
      <ROLE>RoleA1</ROLE>
      <CHILD_SESSION>
       <ACTOR_INSTANCE NAME="A1">
        <ROLE>RoleA11</ROLE>
        <ROLE_SESSION>
         <COOP>ActorA</COOP>
        </ROLE_SESSION>
       </ACTOR_INSTANCE>
       <ACTOR_INSTANCE NAME="A2">
        <ROLE>RoleA12</ROLE>
        <ROLE_SESSION>
         <COOP>ActorA</COOP>
        </ROLE_SESSION>
       </ACTOR_INSTANCE>
       <ACTOR_INSTANCE NAME="A3">
        <ROLE>RoleA13</ROLE>
        <ROLE_SESSION>
         <COOP>ActorA</COOP>
        </ROLE_SESSION>
       </ACTOR_INSTANCE>
      </CHILD_SESSION>
     </ACTOR_INSTANCE>
    </PROPERTY>
   </USER_SESSION>
  </ USER_SESSION_BASE>
```

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. *UserAgent* or *VisitorAgent* is responsible for obtaining and updating the user related information from the director, from the *UserProfileBase*. 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

Table 2. General XML serialization of UserProfileBase

```
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>
      </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>
```

6. ACTOR MOBILITY

Actor mobility stands for the movement of instantiated functionality at a node along its properties, such as behavior, 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. The actor object, as described by TAPAS basic architecture is defined by its: interfaces or role-sessions, behavior, capabilities, queue of incoming requests, and methods accessible by other actors at specific interfaces. Actor Mobility is achieved by re-instantiating actors with their parts. A move procedure is equivalent to a sequence of *ActorPlugIn*, *CapabilityChange*, *CreateInterface*, *BehaviourChange,* and *ActorPlugOut* procedures, which are part of the basic architecture, and used to instantiate an actor, update its capabilities, set a role-session with another actor, change the manuscript it executes, and destroy it, respectively. *ActorMove* request supplies the new location where the actor should be plugged in, while its interface, behavior, capability, queue, and method definition should be preserved from its running instance.

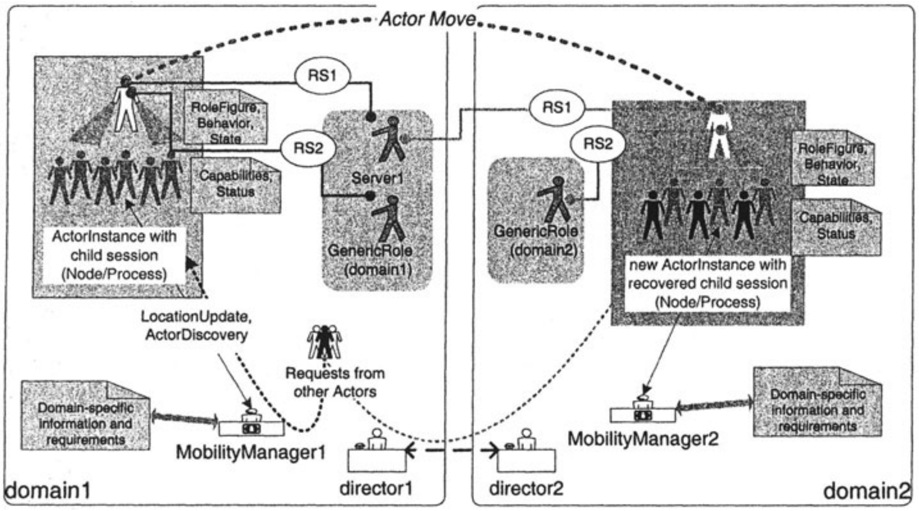
Actor Move
RS1
RS2
RoleFigure, Behavior, State
Capabilities, Status
Server1
GenericRole (domain1)
ActorInstance with child session (Node/Process)
LocationUpdate, ActorDiscovery
Requests from other Actors
Domain-specific information and requirements
MobilityManager1
director1
domain1
RS1
RS2
GenericRole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

Figure 6. Actor Mobility in TAPAS

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 are recovered through applying *CapabilityChange* and *CreateInterface*, B) Behavior part and state information are transferred using

BehaviourChange, and C) queue and method parts are dismissed. Figure 6 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 *GenericRole* (domain1). By moving actors certain parts may be irrecoverable, e.g. certain capabilities may not be available at the new location, or specific role-sessions are no longer relevant, e.g. *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 colors. When an actor moves from one location to another *MobilitManager* is responsible for managing the accessibility to this actor. The actor sends a *LocationUpdate* upon a move, while other nodes send an *ActorDiscovery* to get its location. There is a set of requirements in every domain. Upon moving to 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 7 illustrates a message sequence of an actor changing its location (from L1 to L2), preserving its capability, interface and behavior definitions.

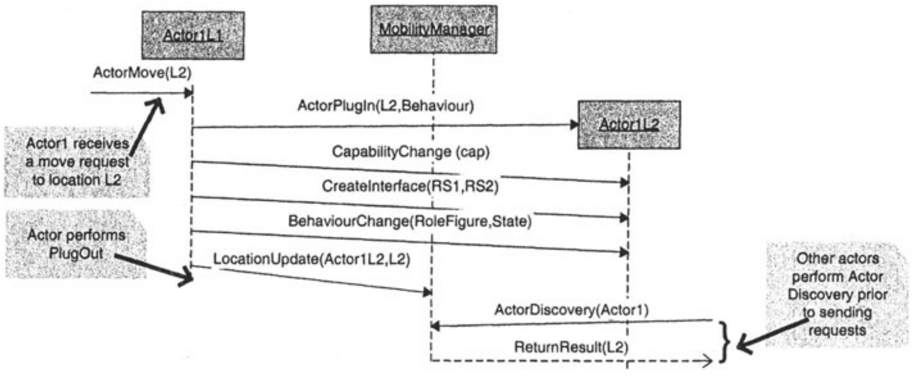
Actor1L1
MobilityManager
Actor1L2
ActorMove(L2)
ActorPlugIn(L2,Behaviour)
CapabilityChange (cap)
CreateInterface(RS1,RS2)
BehaviourChange(RoleFigure,State)
LocationUpdate(Actor1L2,L2)
ActorDiscovery(Actor1)
ReturnResult(L2)
Actor1 receives a move request to location L2
Actor performs PlugOut
Other actors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

Figure 7. Message sequence of a general actor move: "assume Actor1 (Behavior, 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 assigned 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. Terminals execute a *MobilityAgent* responsible for tracking their location, and a manager is responsible for updating the locations of all nodes that participate in any possible 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 location, and *NodeDiscovery* procedures, once a communication is required with other terminals or nodes. Figure 8 demonstrates a general case of terminal mobility, a terminal moves from *doamin1* to *domain2*, while its *MobilityAgent* ensures that *MobilitManager* is updated on this movement. However, when it reaches 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 the domain. *MobilityManagers* operate according to a set of domain specific set of requirements, which govern the privileges and access rights specific user or terminal may have. Figure 9 illustrates a message sequence of terminal mobility.

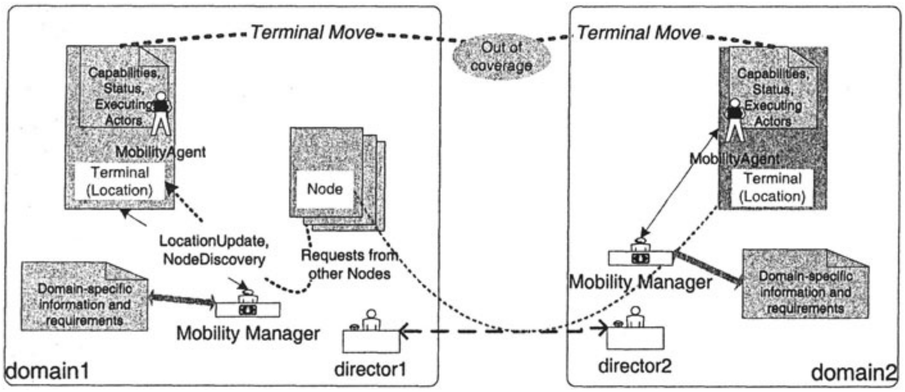
Terminal Move
Out of coverage
Terminal Move
Capabilities, Status, Executing Actors
MobilityAgent
Terminal (Location)
Node
LocationUpdate, NodeDiscovery
Requests from other Nodes
Domain-specific information and requirements
Mobility Manager
director1
domain1
Capabilities, Status, Executing Actors
MobilityAgent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

Figure 8. Terminal Mobility in TAPAS

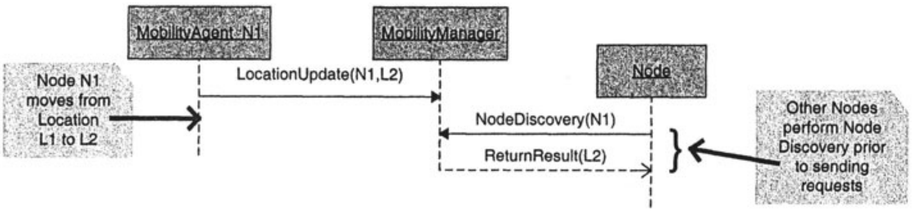
MobilityAgent N1
MobilityManager
Node
Node N1 moves from Location L1 to L2
LocationUpdate(N1,L2)
NodeDiscovery(N1)
ReturnResult(L2)
Other Nodes perform Node Discovery prior to sending requests

Figure 9. 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. *Personal mobility* has 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. 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. Therefore any loss of connection must be recovered, and consequences must be taken into consideration, e.g. marking unconnected actors and deleting their role-sessions. Also, for practical reasons, separate configuration information on the operating wireless environment and user behavior 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 realization 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 formalized 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, to recover the behavior when loosing connections.

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