On Adaptable Networking

Finn Arve Aagesen, Bjarne E. Helvik, Chutiporn Anutariya and Mazen Malek Shiaa
Department of Telematics (ITEM)
Norwegian University of Science and Technology (NTNU)
N-7491 Trondheim, Norway
{finn.arve.aagesen, bjarne.e.helvik, chutiporn.anutariya, mazen.malek.shiaa}@item.ntnu.no

Abstract

Adaptable networking means that the provided network-based services are capable of handling dynamic changes in both time and position related to resources, users and changed service requirements. This paper is discussing several aspects of adaptability. Adaptability implies flexibility, and comprises features related to various aspects of the functionality of a network-based service system such as: 1) software flexibility, adaptability and mobility, 2) personal mobility, 3) dynamic configuration of resources and service software, and 4) flexibility in the interoperating with other architectures. These aspects are discussed with basis in TAPAS (Telematics Architecture for Plug-and-Play Systems).

1. Introduction

Network-based services have during more than one decade been is an important research topic. Example topics include Intelligent Networks [1], TINA (Tele-communication Information Networking Architecture) [2], Mobile Agents and Active and Programmable Networks [3,4,5,6]. Focus has been on service architecture solutions that give flexibility and efficiency in the definition, deployment and execution of the services. This focus is now slightly changing into focus on adaptability and evolution of such services. An example that demonstrates this is The IBM autonomic computing project (http://www.research.ibm.com/autonomic). This new research focus is generally caused from development trends such as:

- increased networking complexity and heterogeneity,
- increased number of parties,
- omnipresent computing and communication,
- decreased hardware and transmission cost,
- increased operation and management workload and cost.

The TAPAS project (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. See [7,8,9,10,11,12] and the URL: http://www.item.ntnu.no/ ~plugandplay.

Another objective is to gain experiences and knowledge by implementing those various features, both for demonstrating the implementation possibility and for validating the feature applicability. The goal is not to develop a complete executing architecture, but is to set the various features coming from the above defined requirements in a context related to totality.

The TAPAS architecture requires a support system for software development, deployment, execution and management. Moreover, the support is also needed for generic user functionality to enable the flexibility features of the system. This support system is denoted as the *TAPAS platform*. Parts of the specified support functionality have been implemented using JAVA RMI and Web technologies as a means for service definition, update and discovery. New versions of the TAPAS platform will use XML as a common representation language.

This paper gives an overview of the status and prospect of the TAPAS architecture and platform, but with focus on the architecture. Section 2 presents the TAPAS basic architecture, which presently is supported by the TAPAS platform. Section 3 presents the mobility handling architecture. Section 4 describes the developed XML-based dynamic configuration architecture, which is not yet implemented as part of the platform. Section 5 presents an infrastructure which enables multiple cross-platform interoperation by employment of Semantic Web technologies.

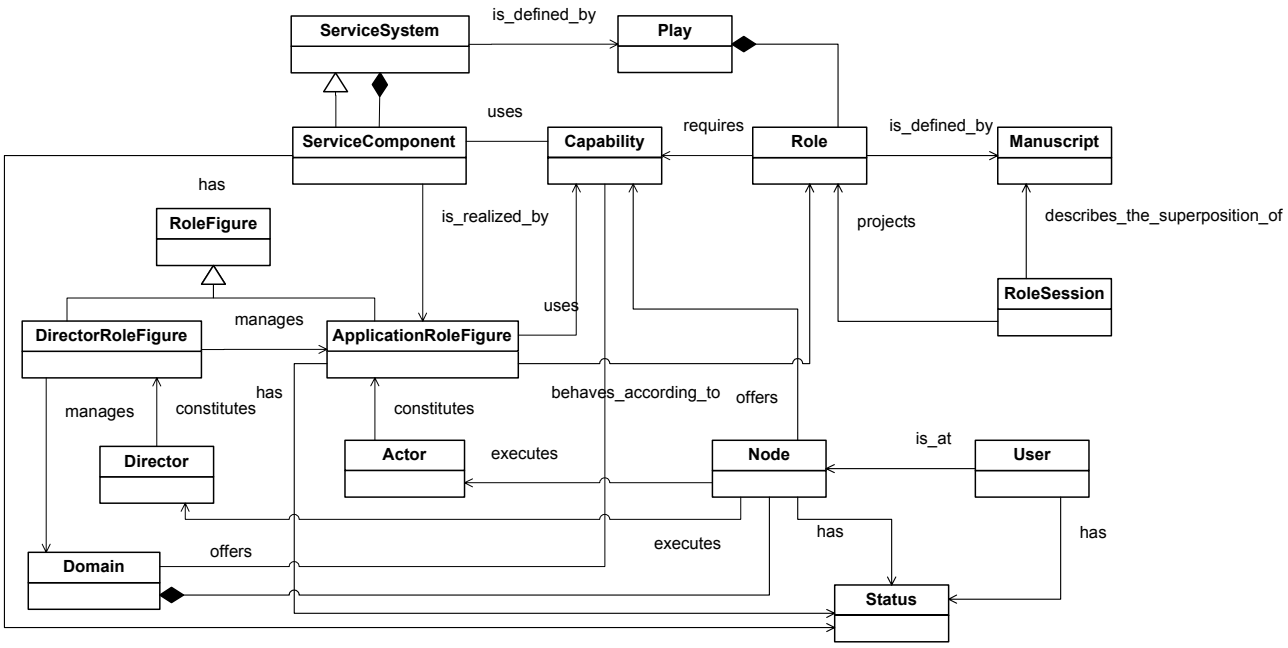

Figure 1. TAPAS basic architecture (Object model)

2. TAPAS basic architecture

The TAPAS basic architecture, illustrated in Figure 1, is based on generic actors in the nodes of the network that can download manuscripts defining roles to be played [8]. These nodes are network processing components, such as 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. Actors are software components, which represent functionality to be executed at different nodes within the network. Roles are modelled as *Extended Finite State Machines*. A director is an actor with supervisory status regarding all other actors' plug-in and plug-out phases. A director also represents a domain, which is a set of nodes managed by a single director.

A service system consists of service components, 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 *role-session* is a projection of the behaviour of an actor with respect to one of its interacting actors. An actor is a generic object, which will constitute a *role figure* by behaving according to a *manuscript* defining the *functional behaviour* of that particular role in a play. A service component is realised by a *role figure* based on a *role* defined by a *manuscript*. A role figure, however, is realised in an executing environment in a node and is utilising capabilities. A capability is an inherent property of a *node*. A node may have several capabilities. These capabilities are offered to actors, which constitute role-figures in various plays. The ability to play roles depends on the *defined required capability* and the *matching offered capability* in a node where an actor is going to play. Examples of capabilities are *processing, storage and communication resources* (e.g., CPU, hard disk and transmission channels), *standard equipment* (e.g., printers and media handling devices), *special equipment* (e.g., encrypting devices), and *data* (e.g., user login and access rights).

A short description of the support functionality for the TAPAS basic architecture is found in [8]. For details see [13,14,15]. Figure 2 gives an example, which illustrates the structure of the support functionality. The Actor-environment-execution-module (AEEM) is a process or thread that executes a collection of actors with associated Plug-and-Play Actor Support (PAS). A collection of actors is here one or more actors constituting application role-figures or director role-figures. The TAPAS platform basic functions supported are provided by the procedures: *PlayPlugIn, PlayChangesPlugIn, PlayPlugOut, ActorPlugIn, ActorPlugOut, ActorBehaviourPlugIn, ActorChangeBehaviour, ActorBehaviourPlugOut, RoleSessionAction*, *ChangeActorCapabilities* and *Subscribe.*

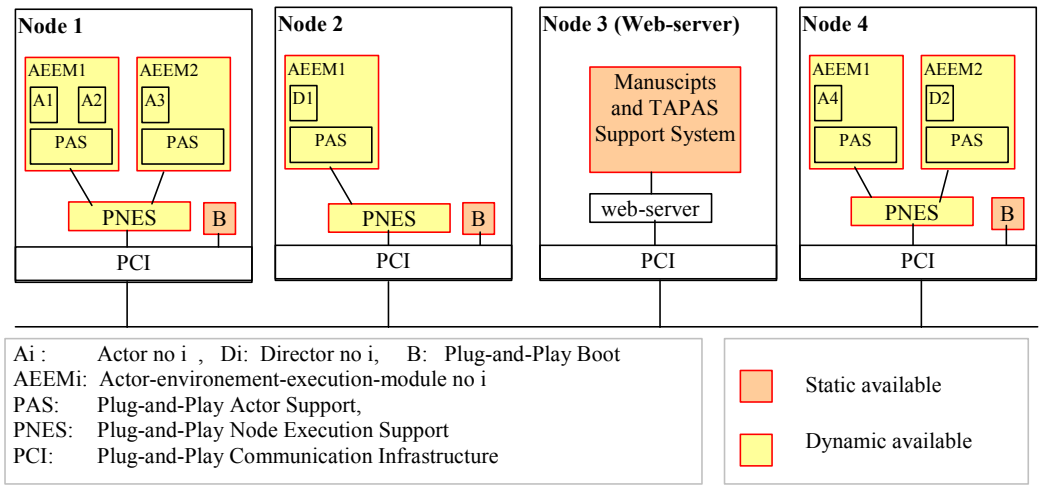

Figure 2. Example view of TAPAS platform for software execution

3. TAPAS Mobility Handling Architecture

3.1. General

TAPAS will comprise four basic mobility features: *user*, *user session*, *actor* and *terminal mobility* [9]. A *terminal* is a node operated by a user. *Terminal mobility* is the physical movement of a terminal. Terminal mobility 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 node capabilities.

Actor mobility is the movement of actors including its role sessions, state and variables. *User mobility* is physical change of the access point of a user, while *user session mobility* is the movement of the user sessions used by one user from one access point to another access point. These four categories is the consequence of the attempts to fulfil the general TAPAS flexibility requirement and to support personal mobility. *Personal mobility*, which will be discussed in Section 3.2, is both related to user, user session, terminal and actor mobility.

Figure 3 presents a model to be used to differentiate and to relate these different types of mobility. A user, 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 object*) within the architecture. This double interface approach (User Interface (*UI*) and Terminal Interface (*TI*)) 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.

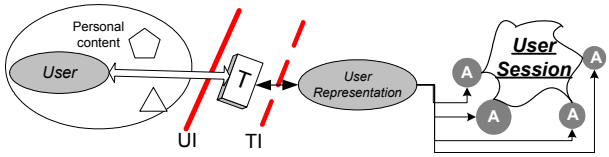

Figure 3. TAPAS basic concept of Mobility

Solutions to all four mobility features are discussed in [9], and recommended solutions are also given. Figure 4 shows an extension of the TAPAS basic architecture illustrated in Figure 1, with emphasis on mobility. These new mobility related concepts will be explained in the next Section 3.2.

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 [11,12]. The present actor realisation based on JAVA RMI does only give a simplified *Actor mobility*. However, a new and more powerful actor model is being developed. In addition to being an Extended Finite State Machine, the actor will have methods that can be activated. The state of the actor can be made available by activating such a method.

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. This mobility handling functionality will be extended to reason about terminal capabilities and status of the networking environment based on the methodology of the architecture in Section 4.

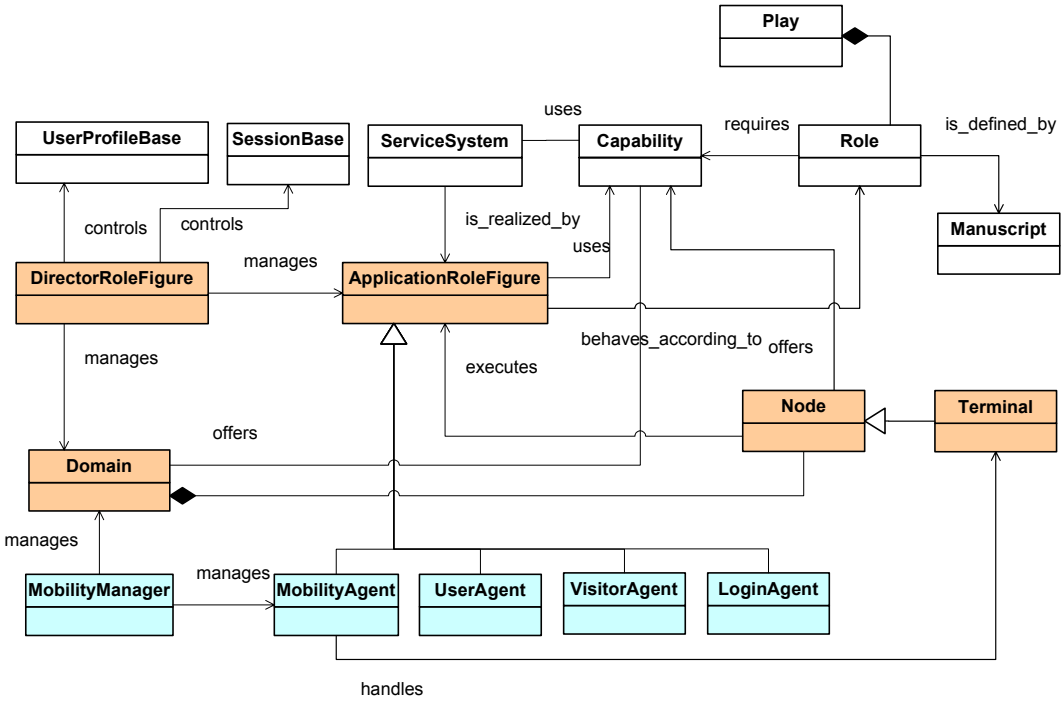

Figure 4. Complete view of mobility handling within TAPAS (object model)

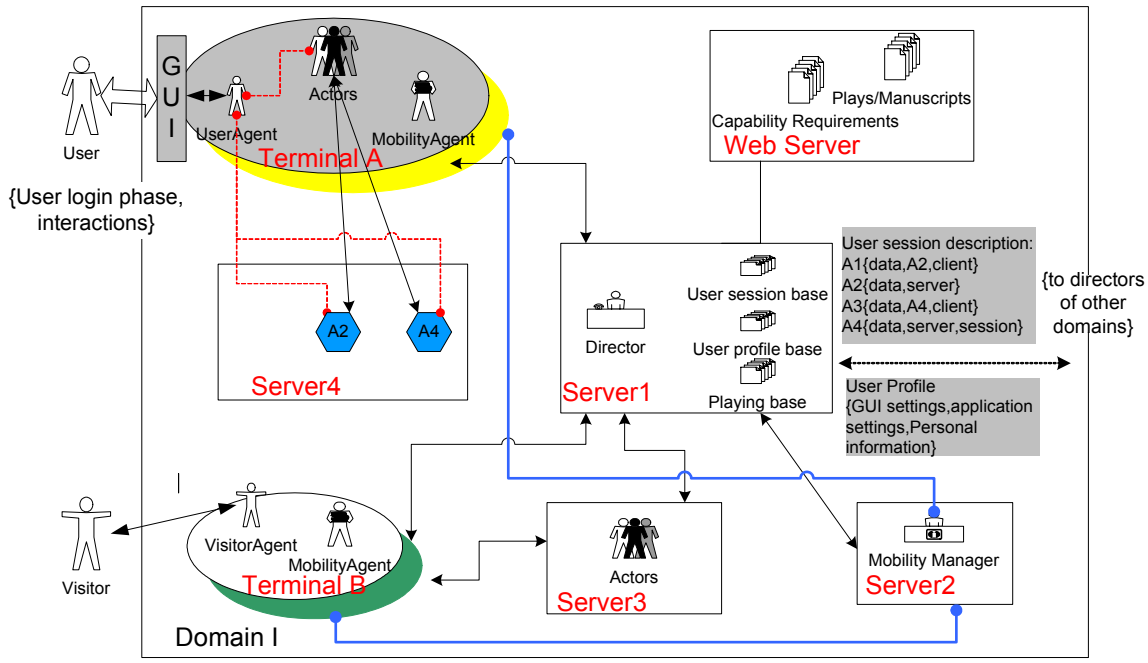

Figure 5. An illustration of TAPAS mobility framework

3.2. Personal Mobility

Personal mobility can be defined by: "the utilization of services that are personalized with end user's preferences and identities independently of both physical location and specific equipment". Figure 5 illustrates the solution to *Personal mobility* in TAPAS:

- A *User* is referred to by an ID and *User Profile,* that is active through a *User Interface* (or *GUI*). Additionally, user-to-terminal relation is defined at *login phase*. Director maintains *User Profiles,* which might contain information on user settings, preferences and personal data.
- *Interactions between users and application actors* are controlled by a *UserAgent*, which is unique for one user session.
- *Personal content* is defined by user applications and user profiles.
- *Terminals* and *nodes* are characterized by different set of capabilities. Thus, certain application components run at network nodes instead of user terminals.
- *UserAgent* is keeping track of any actor created or used in a User session.
- *Multi-domain* environment is achieved by allowing the domain's director, one director in one domain, to contact other directors inquiring about *visitor* user's ID and profile. Visitors are assigned a *VisitorAgent* to control their interactions while they are in the system. Two types of login are defined: local and visiting.
- User sessions are maintained by the director in the *User Session Base,* that contains *user session descriptions*. These are detailed sketch of running services, actors and their related data.

Figure 5 also illustrates how a user session is managed by the *UserAgent*, and consequently maintained by the director's data base. The connectors between the Actors in *Terminal A* and the actors *A2* and *A4* in *Server 4* show that they are inter-related and correspond to the same user. The dotted connectors from *UserAgent* to these actors represent one user session. An example is provided for a *session description* and a *user profile*. In this example, actors are distributed on the user's terminal and a network node; a typical example is a chat client and a server. When a user session is suspended, information on every actor's data, e.g. user name, connections, type of application and information for child sessions should be stored. User's login phase is central to the definition of user identity, characterisation of device capabilities, resumption of user sessions, and transfer of personal contents. As mentioned earlier, users can have either local or visiting login. However, visitor users can access their home domain if inter-director negotiation and authentication is possible with their home domain.

4. TAPAS Architecture for Dynamic Configuration Handling

Due to dynamic availability of nodes in the network as well as changes in their capabilities and status, configuration and reconfiguration of nodes to constitute particular service components must not be predetermined but be computed on the fly. To deal with such a requirement, Figure 6 illustrates the extended TAPAS architecture for dynamic (re)configuration of plug-and-play (PaP) systems, which comprises the following primary elements:

1. *Capability & Status Repository* (*CSRep*) maintains *capability* and *status* information of components in the system. Capability information characterises inherent properties of each component and is classified into resources, functions and data, while status information reflects the situation of a PaP system at a particular time, which can be, for instance, certain environment conditions, observable values of the current QoS characteristics as well as their calculated measures. With an emphasis on the use of a standard schema for modelling capabilities and status, the developed architecture uses and extends CIM (Common Information Model) [16]—a fundamental yet comprehensive object-oriented schema for describing network resources in XML format. Based on this modelling concept, capability and status information of a particular component is modelled as a corresponding CIM instance, and the CSRep is then represented as a collection of CIM instances which together describe the available capabilities and status of the components in the running PaP system. This capability and status information is typically analysed by the Configuration Manager when computing (re)configuration plans for the system.
2. *Play Repository* (*PlayRep*) stores a collection of *play definitions*, each of which defines requirements and functional behaviours of a corresponding PaP service system. In particular, a play definition is an aggregation of the four specifications:
   (i) *Manuscripts* define the entire functional behaviour of each role in a play which not only includes its internal behaviour, but also the interactions and cooperation with other roles.
   (ii) *Role specifications* identify the capability and status requirements of each role.
   (iii) *Play configuration rules* describe system configuration constraints which must always be maintained, such as the maximum number of roles allowed to install at a specific node in order to avoid an overload situation, the desired or acceptable QoS levels of the system.

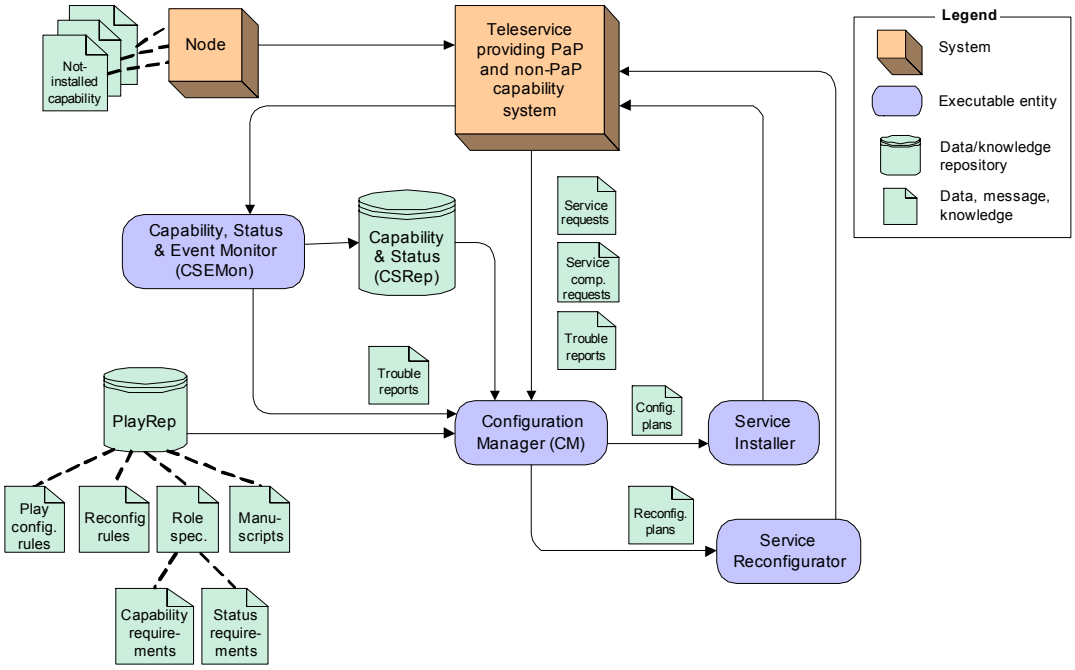

Figure 6. TAPAS dynamic configuration architecture

(iv) *Reconfiguration rules* define application-specific reconfiguration policies for handling significant reconfiguration-related events, such as a service component failure, a decrease in system QoS and resource unavailability. Instead of providing merely a general reconfiguration mechanism, which is applicable to any trouble encountered in an application. These reconfiguration rules let different applications encode their individual, customised policies, and hence allowing them to handle the same trouble in different but application-specific manners.

While a manuscript is specified by an EFSM (Extended Finite State Machine), a role specification, play configuration rule and reconfiguration rule are uniformly formalised within a single representation schema, i.e., XML Declarative Description (XDD) [17,18], as a corresponding XDD description.

3. *Data Messages*, encoded in RDF (Resource Description Framework) [19,20]—the W3C recommended metadata and the Semantic Web [21] language—provide means to communicate among various entities in the architecture. Basically, each message carries its URI (Universal Resource Identifier), information of the actor who sends the message (i.e., the sender) and the date/time of composing it. A sender's information includes its URI, the installing location and the playing role. Other message attributes can also be encoded depending on the purpose of the message. In the architecture, messages are classified into: requests, trouble reports, configuration plans and reconfiguration plans.

(i) *Requests* are further divided into: a service request and a service component request. The former is a request for installation and execution of a particular PaP service system, which has not yet been installed in a PaP system. It encodes the service request URI, the requester information, date/time, and the requested play URI identifying the type and version of the service to be installed. The latter is a request for instantiation of a particular service component in a running PaP service system, which contains the request URI, information of the actor who makes the request, date/time and the name of the role for realising the desired component.

(ii) *Trouble reports* play a vital role in the architecture by providing each component in a running system as well as the Capability, Status & Event Monitor with the ability to report a problem that demands an immediate system adaptation to the Configuration Manager. Examples of trouble reports are actor unreachable report, insufficient capability report and QoS degradation report.

(iii) *Configuration plans* consist of lists of appropriate locations for initialising actors and

installing manuscripts of each specific role in a given play version.

(iv) *Reconfiguration plans* are generated by the Configuration Manager in order to cope with certain troubles occurred in a system. Possible plans include: no action, actor initialisation, actor termination, actor re-initialisation, actor relocation and play reconfiguration.

4. *Capability, Status & Event Monitor* (*CSEMon*) monitors PaP system capabilities/status and maintains the CSRep. Moreover, it listens to certain events indicating changes to the system and its environment, which would prevent the system from getting the desired level of *services*. In response to such events, it notifies the Configuration Manager for further proper reactions, in order to keep the system functioning with an acceptable QoS level.
5. *Configuration Manager* (*CM*) is responsible for:

(i) Generation of appropriate configuration plans for composing new services:

In response to a given service request, the CM fetches a corresponding play definition from the PlayRep and retrieves the system capabilities and status from the CSRep. Valid configuration plans for such a service are then computed, and an appropriate one will be selected based on the specified selection criteria, e.g., system performance and QoS, user preferences and cost. The selected configuration, defining which nodes in the system should execute actors constituting certain roles, will be forwarded to and executed by the Service Installer.

(ii) Determination of a location for executing a particular role:

In the running PaP service system, a request for instantiation of a particular service component, i.e., a service component request, may arise. In response to such a request, the CM dynamically determines the best location (node) for its installation, based on the current system configuration, available capabilities and status as well as the component's requirements. It then notifies the Service Installer to load a corresponding manuscript from the PlayRep and instantiate it on the suggested node.

(iii) Computation of reconfiguration plans for dynamic reconfiguration of the executing services:

Upon the receipt of a trouble report indicating a problem in a running PaP system, the CM analyses the problem, fetches related information from the CSRep and the PlayRep, and produces a service reconfiguration plan to be executed by the Service Reconfigurator. Selection of an appropriate plan depends on the defined reconfiguration rules, their priority information as well as the nature of a problem (e.g., whether it is hardware or software failure, significant or ignorable).

6. *Service Installer* is responsible for the installation of a service into the PaP system by creating corresponding actors for execution of certain roles, according to an obtained play configuration generated by the CM. Allocation of capabilities as well as instantiation of a manuscript for each role are also performed by this entity.
7. *Service Reconfigurator* initiates and performs reconfiguration of a service system based on an obtained *reconfiguration* plan.

It is seen from the architecture that the Configuration Manager is the primary entity which dynamically computes appropriate service (re)configuration plans by reasoning about the current system's capabilities & status, the defined role requirements, play configuration constraints and reconfiguration rules as well as the given requests and trouble reports. This mechanism allows, for a particular service installation, deployment and execution, a variety of (re)configuration policies to be defined in a customisable, domain-specific manner, each possibly resulting in different configuration and reconfiguration plans, and hence enabling the system to cope with variations in the environment, achieve mandated performance levels and meet user satisfaction. The implementation of the architecture using an effective XML-based reasoning engine and its integration into the TAPAS platform is underway.

5. Adaptive Service Infrastructure

In such a global-scale, heterogeneous network environment as the Web, services are increasingly complex and diverse in terms of, for example, availability, capabilities, platforms and technologies. Moreover, the direction of today's computing is decomposition of a highly integrated computing system into a collection of heterogeneous, distributed and fragmented systems, possibly implemented using different platforms and technologies and often operated by different providers. This trend enables a service provider to construct a higher-level service from the composition of multiple lower-level services (or sub-services), instead of implementing the service as a whole brand-new dedicated, specific software application. The realisation of this new trend demands a well-established infrastructure which provides a set of well-defined interfaces and enables dynamic and cross-platform composition, instantiation and interoperation of heterogeneous, adaptive services regardless of their

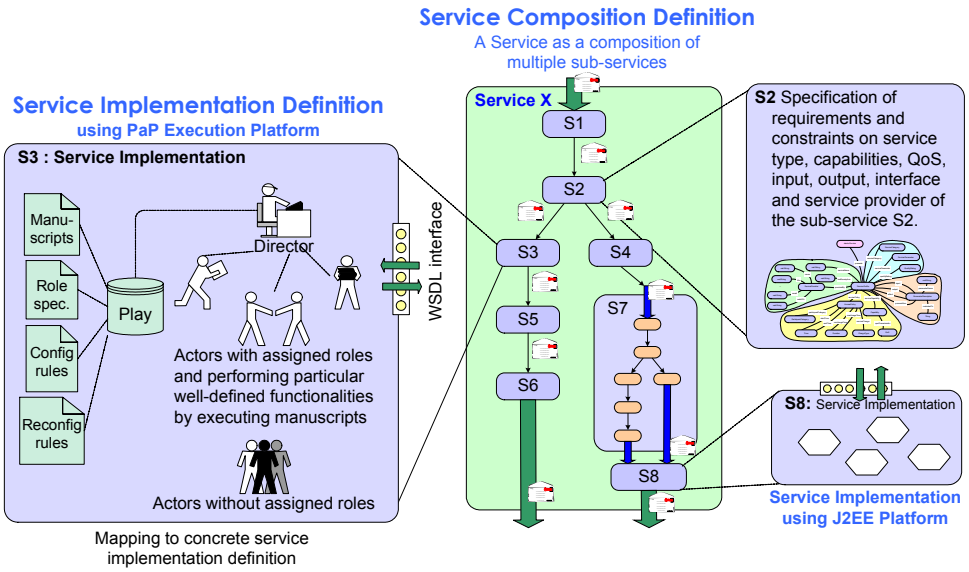

Figure 7. Example of a service composition definition

programming languages and operating environments. This section presents such an infrastructure by adopting the emerging Semantic Web [21] and Web services technologies, and discusses also how the TAPAS architecture fits into the big picture.

5.1. Service Definition

A *service definition* is a specification of how a service is to be realised. Such a service can be atomic (i.e., a non-decomposable service) or a composition of multiple sub-services related to each other by certain control flows and data flows. This composition structure can be static or dynamic. In the former case, selection of sub-service instances and their providers are predefined, while in the latter case, services are assembled dynamically and the processes of discovering appropriate sub-service instances and their providers are determined at runtime.

Here, WSFL (Web Services Flow Language) [22] is employed to describe the composition structure as well as the control flow and data flow of a composite service. The concept ServiceProfile in DAML-S (DARPA Agent Markup Language for Web Services) [23,24] is extended with facilities to express the requirements on properties, capabilities and QoS constraints of each sub-service and of its provider. In addition, in order to enable dynamic, automatic service invocation and interaction, WSDL (Web Service Description Language) [25] is used to define service interface definitions and access bindings.

Figure 7 presents an example of a dynamic service composition definition, which specifies that a service X is a composition of sub-services S1, S2, …, S8, where the input of the service X will be directed to the sub-service S1, the output of which will be sent as an input to S2, etc. In this example, it does not specify how the sub-service S2 is implemented or realised, instead it merely defines certain desired properties, capabilities, QoS, input, output, interfaces or costs of S2, hence allowing dynamic discovery and binding of an appropriate, available service that matches the need. The sub-services S3 and S8, on the other hand, are defined statically by mapping to particular service implementations, which execute under the TAPAS and J2EE platforms, respectively. The definition of the sub-service S7 shows that a sub-service can also be further decomposed into a set of other sub-services. Moreover, the example explicitly illustrates that each (sub-)service provides a WSDL interface in order to enable a standard invocation and interaction mechanism among these independently developed and cross-platform services.

5.2. Service Advertisement and Discovery

To facilitate automatic discovery of available services, their semantic descriptions should be described in an unambiguous and machine-comprehensible manner and be advertised in certain registry services, where other participants can query and search for services that provide a set of desired capabilities. In the developed infrastructure, RDF [19,20] and DAML-S [23,24] languages are employed and extended with facilities for describing service capabilities, QoS and access policies, which enable a service provider to, for example, restrict a service to a particular group of participants with certain access time, minimum/maximum service usage duration and type of service charge, such as free-of-charge, lump-

sum-based or duration-based. In addition, using available registry services, a service consumer can browse through the service category ontology and search for a service provider which offers a service that best matches his/her requirements.

5.3. Service Negotiation, Selection and Service Contract

Given a service requirement, there might exist various available services which deliver the desired functionality with similar capabilities, but differ in terms of service qualities, prices and providers. The criteria for selecting a service may vary depending on the nature of the service, the policies, and preferences of a consumer (e.g., the cheapest, fastest, most accurate, or a trade-off among these price-performance issues). Moreover, it is possible that no available service can fulfil the consumer's need. Thus, some form of automated negotiation, usually based on various types of auctions, is required. After the service provider and consumer both come to an acceptable agreement, a service contract—specifying, for example, what the provider should deliver, the guaranteed QoS and the cost—is established.

5.4. Dynamic Service Composition, Instantiation and Adaptation

After having set up a service contract, the provider instantiates and invokes the service by dynamically assembling a set of related sub-services according to the service definition and the service contract. Selection and instantiation of each sub-service can be an iterative of the overall process because it can again involve discovering an appropriate provider, negotiating, setting up a contract and invoking each sub-service.

In addition, to ensure that the terms, conditions and the service quality as specified in the contract are maintained, the executing service must be monitored and certain appropriate adaptation performed when needed. To enable this, the TAPAS platform for realising and delivering services together with its architecture for dynamic configuration and adaptation can be employed.

Upon the completion of the service, the output is delivered to the consumer in form of either a simple message informing the service completion or some complex contents representing the results of service execution. The executing service is then terminated.

6. Conclusions

Adaptable networking means that the provided network-based services are capable of handling dynamic changes in both time and position related to resources, users and changed service requirements. This paper has been discussing several aspects of adaptability, and an architectural concept TAPAS (Telematics Architecture for Plug-and-Play Systems) has been presented. This is the result of a research project which aims at designing 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. The objective of this work is to simplify and speed up the tasks of deployment, installation, operation, management, maintenance and evolution of software related to telecommunication equipments and services

The TAPAS architecture was presented as four architectural concepts: 1) the basic architecture, 2) the mobility handling architecture, 3) the dynamic configuration architecture, and 4) the adaptive service infrastructure. The basic architecture is the basis for all dynamic behaviour functionality, which is based on generic actors in the nodes of the network that can download manuscripts defining roles to be played. The model is founded on a theatre metaphor, where actors perform roles according to predefined manuscripts, and a director manages their performance. The roles are modelled as Extended Finite State Machines. The mobility handling architecture and the architecture for dynamic configuration is based on the basic architecture, but is also enhancing the functionality in the basic architecture by functionality to meet the requirements A)-B) as defined above. The fourth architectural concept is different. Assuming that there never will be only one and only service architecture. How can different architectures interoperate with each other? A solution based on Semantic Web was proposed.

There are basically two different approaches for meeting adaptability and flexibility requirements. The philosophy selected here is to use a system that has knowledge and overview, and to make this system robust and survivable. An opposite approach is architectures based on Swarm Intelligence [26], by using intelligent moving agents and to apply simple biological models for the behaviour. The TAPAS actor can move, but the move as well as the behaviour is part of a defined service functionality. Solving adaptability and flexibility creates complexity. May be there are easy solutions some place out there. We do, however, propose a solution where internal platform complexity, which by nature itself is flexible and adaptable, can be the fundament for adaptable and flexible service

functionality. Adaptability and flexibility must reside on any level of the architecture. From the Actor-to-Actor level to the Platform-to-Platform level. In between these levels, we must meet the flexibility and adaptability requirements of users, services and capabilities.

The basic architecture features as well as most of the mobility handling architecture have been implemented and validated. Further work will be on the basic actor model, the capability handling architecture as well as the interoperating architecture. The basic actor will be a generalized Extended Finite State Machine with methods that can be activated from outside. This is needed to handle appropriate movements of Actor states and role-sessions. The work on the capability handling architecture comprises the use of an effective XML-based reasoning engine and its integration into the TAPAS platform. The work on the interoperating architecture using ideas from the Semantic Web technologies is in its starting phase.

References

1. ITU-T, Principles of intelligent network architecture, October 1992.
2. Inoue, Y., Lapierre, M. and Mossotto, C., *The TINA Book: A Co-operative Solution for a Competitive World*, Prentice Hall, 1999.
3. Bieszczad A. and Pagurek B., "Towards Plug- and Play Networks with Mobile Code", *Proc. ICCC'97*, November 1997.
4. Bieszczad A., Pagurek B. and White T., "Mobile Agents for Network Management", *IEEE Communications Surveys*, Vol. 1, No. 1, 1998.
5. Raza S.K. and Bieszczad A., "Network Configuration with Plug and Play Components", *Proc. 6th IFIP/IEEE International Symposium on Integrated Network Management*.
6. 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.
7. 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.
8. Aagesen, F. A., Helvik, B.E., Johansen, U. and Meling, H., "Plug and Play for Telecommunication Functionality: Architecture and Demonstration Issues", *Proc. Int'l Conf. Information Technology for the New Millennium* (*IConIT*), Thammasat Universit, Bangkok, Thailand, May 2001.
9. 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.
10. 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,
11. 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*, Trondheim, Norway, September 2002.
12. Shiaa M.M and Aagesen F.A., "Architectural Considerations for Personal Mobility In the Wireless Internet", *Proc. IFIP TC6/ WG 6.8 Conf. Personal and Wireless Communications (PWC2002)*, Singapore, Kluwer Academic Publishers, October 2002, pp. 285-292.
13. Johansen U., Aagesen F.A., Helvik B.E. and Meling H., "Design Specification of the PaP Support Functionality", *Plug-and-Play Technical Report*, Department of Telematics, NTNU, 1999-12-10, ISSN 1500-3868.
14. Johansen U., Aagesen F.A., Helvik B.E. and Meling H., "Demonstrator - Requirements and Functional Description", *Plug-and-Play Technical Report*, Department of Telematics, NTNU, 1999-12-10, ISSN 1500-3868.
15. Johansen U., "Plug-and-play – Software Design, Implementation and Use", *Plug-and-Play Technical Report*, Department of Telematics, NTNU, 2001-02-10, ISSN 1500-3868.
16. Westerinen, A. and Strassner, J., "Common Information Model (CIM) Core Model, Version 2.4", *DMTF White Paper*, 2000.
17. Wuwongse, V., Akama, K., Anutariya, C. and Nantajeewarawat, E., "A Data Model for XML Databases", *J. Intelligent Information Systems*, Kluwer Academic Publisher, Vol. 20, Issue 1, 2003, pp. 63-80.
18. Wuwongse, V., Anutariya, C., Akama, K. and Nantajeewarawat, E., "XML Declarative Description: A Language for the Semantic Web", *IEEE Intelligent Systems*, Vol. 16, No. 3, May/June 2001, pp. 54–65.
19. Lassila, O. and Swick, R. R., "Resource Description Framework (RDF) Model and Syntax Specification", *W3C Recommendation*, February, 1999.
20. Brickley, D. and Guha, R.V., "RDF Vocabulary Description Language 1.0: RDF Schema 1.0." *W3C Working Draft*, 30 April 2002.
21. Berners-Lee, T., Fischetti, M. and Dertouzos, T.M., Weaving *the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*, Harpur, CA, 1999
22. Leymann, F., "Web Services Flow Language (WSFL 1.0)", *IBM White Paper*, May 2001.
23. Hendler, J. and McGuinness, D., "The DARPA Agent Markup Language", *IEEE Intelligent Systems*, Vol. 15, No. 2, March/April 2000, pp. 72–73.
24. McIlraith, S. A., Son, T. C. and Zeng, H., "Semantic Web Services", *IEEE Intelligent Systems*, Vol. 16, No. 2, March/April 2001, pp. 46–53.
25. Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S., "Web Services Description Language (WSDL) 1.1.", *W3C Note*, March 2001
26. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, "Ant-based Load Balancing in Telecommunications Networks", *Adaptive Behavior*, Vol. 5, No. 2, pp. 169-207, 1996.