XML-based Dynamic Service Behaviour Representation

Shanshan Jiang and Finn Arve Aagesen

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

{Shanshan.Jiang, Finn.Arve.Aagesen}@item.ntnu.no

Abstract

This paper presents an XML-based framework for implementation language and platform independent service execution behaviour representation. It is based on an FSM-interpreter which can interpret and execute XML-represented FSM behaviour. This model has been integrated into TAPAS (*Telematics Architecture for Plug-and-Play Systems*) platform. TAPAS has basically two engineering viewpoint functionality classes: i) Dynamic service configuration and ii) Dynamic service instantiation. The dynamic service configuration is already based on XML. Using XML for the service execution representation gives one integrated representation of all dynamic service related functionality. As XML basically is a structure representation language, dynamic behaviour representation is made possible by utilising the inherent characteristics of the TAPAS basic architecture. TAPAS defines the behaviour by plug- and- play manuscripts and Node inherent capabilities.

Keywords: XML, EFSM, Plug-and-Play, service behaviour description

1. Introduction

Due to the increase in both heterogeneity and complexity in today's networking, wide-area distributed computing and service provision systems, there arises a demand for an architecture for network-based service systems that gives flexibility and efficiency in the definition, deployment and execution of the services and at the same time, takes care of the adaptability and evolution of such services.

The *TAPAS* project (TAPAS = *Telematics Architecture for Plug-and-Play Systems*) [1,2,3,4] 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.

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 consists of two service *functionality classes*: P) the basic

architecture and Q) the mobility handling architecture, where Q) is an extension of P). Considering the functionality from an engineering viewpoint, the *engineering functionality* can be classified as i) Dynamic service configuration and ii) Dynamic service instantiation. *Dynamic service configuration* makes decision about in which Node to run the service software. This decision is based on information about required and offered status and capabilities. *Dynamic service instantiation* comprises service creation, definition, deployment, execution and management.

The TAPAS architecture requires a support system for the engineering functionality. The support system for dynamic service instantiation is denoted as the TAPAS platform. The Dynamic service configuration functionality is based on the TAPAS platform, and the engineering dynamic configuration functionality can also be considered as a service functionality class R) in addition to the classes P)-Q) described above.

Parts of the specified support functionality have previously been implemented using Java RMI and Web technologies as a means for service definition, update and discovery. For more information of the architecture, implemented system and ongoing research activities, the reader is referred to [1,2,3,4] and the Web site http://tapas.item.ntnu.no/ .

The TAPAS basic architecture is based on actors that can download manuscripts defining roles to be played. 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. XML (eXtensible Markup Language) [6] is a standard for interchanging structured documents over the Internet. It has potential in offering interoperability across heterogeneous systems, and can be used to integrate systems by interchanging data and metadata (data about data). The TAPAS dynamic configuration functionality uses standard XML-based metadata and ontology languages for modelling and providing semantic description of capabilities and status of Plug-and-Play (PaP) systems [2]. By also using XML as a basis for the service instantiation functionality, we will have one common representation language through the whole architecture. This is the motivation for the work described in this paper. A model is presented, which uses XML as a representation language for the basic behaviour of the Extended Finite State Machines (EFSM), but where the actions are based on Node inherent capabilities. This model has been integrated into the TAPAS platform. The model is at the moment basically supporting the basic architecture P), and this paper is also concerned about this basic architecture. Parts of the mobility handling architecture Q) is already XML-based [5], but the solution presented here will be used as the basis for actor movement, which is a needed function for the fully support of all aspects of mobility handling [5].

The paper is organized as the following. Section 2 describes relevant parts from the TAPAS architecture. Section 3 gives the model and framework for XML-based behaviour modelling. Section 4 describes the implementation using Java technology and its integration into TAPAS platform. Conclusions are given in section 5.

2. TAPAS basic architecture and dynamic configuration functionality

TAPAS has already been classified related to service and engineering functionality. This Section is focusing on TAPAS basic architecture and the dynamic configuration functionality.

The TAPAS basic architecture (Figure 1) is based on generic actors in the nodes of the network that can download manuscripts defining roles to be played. This 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 modeled as EFSM. 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 will constitute a *role figure* by behaving according to a *manuscript* defining the *functional behaviour* of a 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* (or a *capability component*). A capability component may have several capabilities. These capabilities are offered to actors. 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). Capability can be specialized into three subclasses: Function, Resource and Data [1]. Functions are pure software or combined software/hardware components used for performing particular tasks.
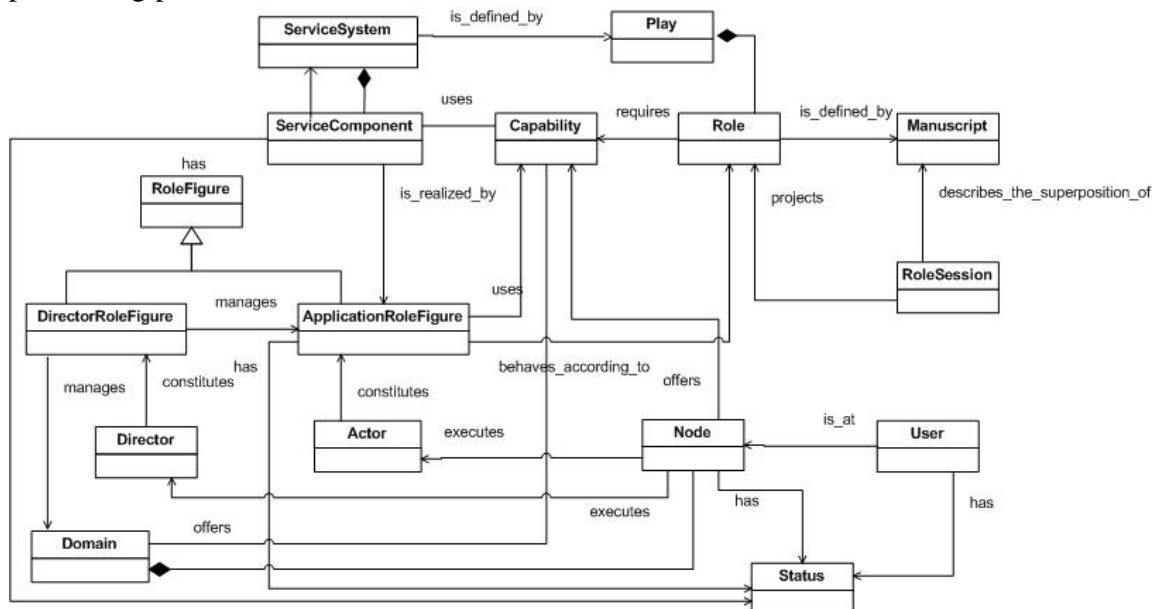
Figure 1. TAPAS basic architecture (object model)

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 be computed on the fly. The TAPAS dynamic configuration functionality (Figure 2) is extended to deal with such a requirement [2]. As seen from Figure 2, the Play repository stores a collection of play definitions, each of which defines requirements and functional behaviours of a corresponding PaP service system. A play definition consists of four specifications: 1) Play configuration rules, 2) Reconfiguration rules, 3) Role specifications and 4) Manuscripts. Role specifications identify the requirements on available capabilities and status for each role. Role specifications and manuscripts define two aspects of roles in a play: the static and dynamic models. The former describes the metadata of each role and the latter models its functional behaviour in terms of EFSM. Hence, they provide information of "what the role is" and "how it is realised", respectively. Role specifications, play configuration and reconfiguration rules are uniformly formalised in a single representation schema, i.e., XML Declarative Description (XDD) [14].
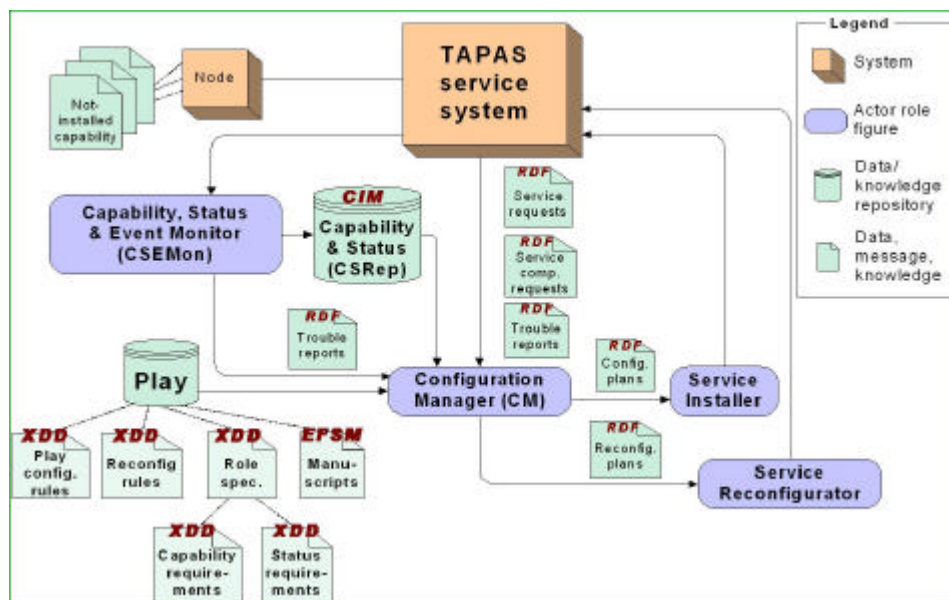

Figure 2. TAPAS dynamic configuration functionality

3. Behaviour description using XML

XML is appropriate for the modelling of structure and data. However, to use XML to represent service logic, i.e., the dynamic behaviour, poses a challenge. Some XML-based programming language are available, such as Superx++[8] and XL[9]. But the limitation for using such a language is that the language is not general enough and not portable.

In TAPAS, the behaviour is defined by plug-and-play manuscripts and Node inherent capabilities. The manuscript definition is basically a state machine specification of certain functionality, and describes the behaviour of the actor performing it. When a manuscript is executed, the internal actions of the state machine can utilise the functions provided by nodes, which are inherent capabilities of the nodes. On the other hand, once a manuscript is

downloaded and a role is plugged-in on a node, the actions this role figure can perform represent new capabilities the node provides.
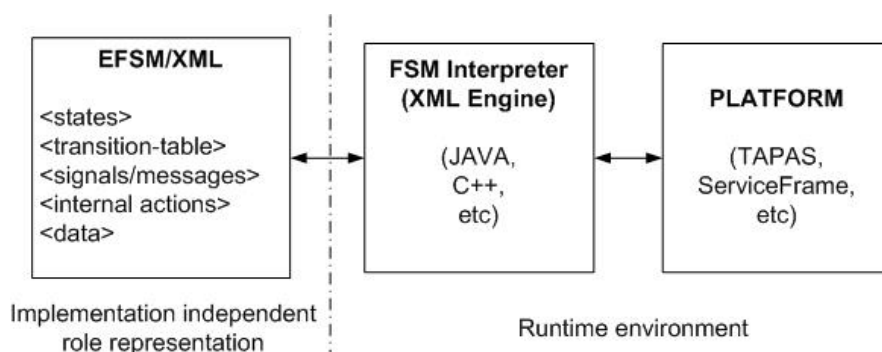

Figure 3. XML-based service system model

The model for representing EFSM using XML is illustrated in Figure 3. FSM interpreter is the core part for this model. It is the interface between XML-based EFSM description and the execution platform. Figure 4 gives the functionality for the FSM interpreter. The FSM interpreter has some basic functionality, which is same for all applications. This includes loading XML file into memory when initializing an FSM, signal/message processing (sending / receiving), internal actions, state transitions and FSM management. In addition, this FSM interpreter can have extended functionality related to different applications. This includes, for example, special algorithms for role negotiation and PaP support functionality (e.g., the use of ActorPlugIn, RoleSessionAction and other APIs). The FSM interpreter can be implemented by Java, C++ and other programming languages depending on the platform used, which may be TAPAS, ServiceFrame [7] and others.
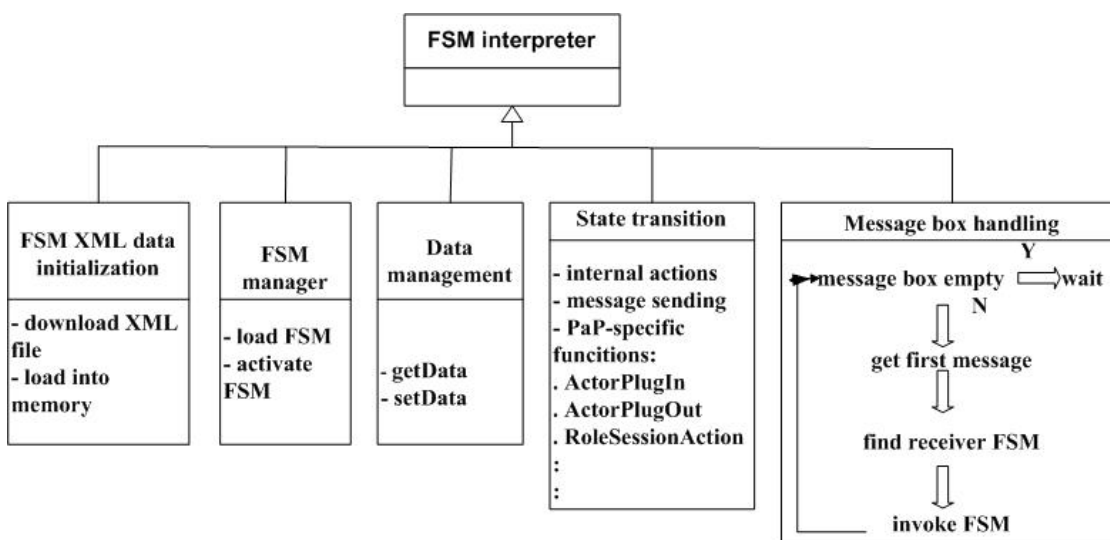
Figure 4. Functionality for FSM interpreter

The XML file has a simple and well-extensible data structure. It is implementation language independent. The XML file consists of all the basic elements for state machines (Figure 5). The set of states is defined together with initial state and current state. The transition table is used to make transitions according to the current state and the input signal/message. Application specific data is also recorded in the XML file. Internal actions that are carried out during a state transition are defined with an <action> list. This XML structure has good extensibility and allows for easily adding new abstractions. For example, if method interface is desirable for the EFSM, the method signatures can be represented in a similar structure as <action> list.
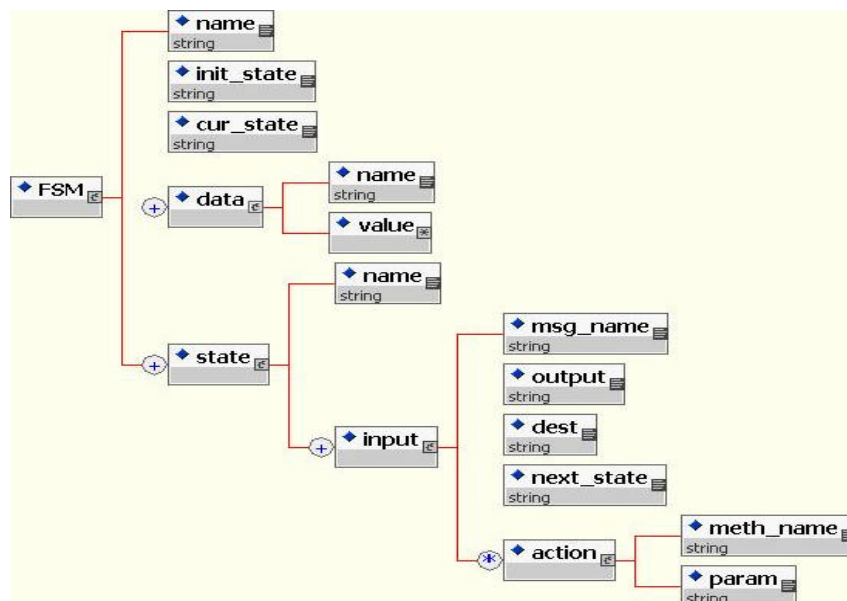

Figure 5. Basic XML data structure

Different FSM Interpreters can be implemented to exchange XML-based behaviour description in different systems. XMI (XML Metadata Interchange) [10,11] is a widely used exchange format, which is also based on XML. XMI is an OMG Standard, and a model driven XML Integration framework for defining, interchanging, manipulating and integrating XML data and objects. "The main purpose of XMI is to enable easy exchange of metadata between modelling tools (based on the OMG UML) and metadata repositories (OMG MOF based) in distributed heterogeneous environments [10]." In short, XMI is UML (Object Technology) with XML (exchanging data over Internet). XMI is currently being adopted by a lot of UML Case Tools vendors and used to tie all tools together. For example, using XMI Toolkit from IBM, developers can share Java objects using XML, generate document type definitions (DTDs), and convert designs and code between Java, UML, and the Rational Rose visual modelling tool [12]. To utilise the benefits from XMI, there can be part of the extended functionality for FSM Interpreter to translate XML file into XMI format and XMI to XML for interchange between tools. Since there is no inherent confliction between XMI and the XML-based EFSM file as they have common basis on XML, this conversion functionality can be easily implemented.

4. The implementation in Java and TAPAS platform

An example for this XML-based behaviour representation has been implemented on TAPAS platform using Java technology. The reason for using Java is that Java has many strong points for XML development. It is a mature technology, highly portable and supported by many vendors. Many XML tools and high-quality development environments are also available in Java.

The implementation framework is given in Figure 6. The manuscripts are defined in XML. An FSM interpreter is written in Java to interpret the XML files to work with the Java-based TAPAS platform. The role figures are represented as FSM instances. The FSM interpreter manages all the FSM instances in the memory and handles a common message box for all these FSMs. The manuscripts in XML are downloaded from the Web server during plug-in phase and loaded into memory as FSM instance by FSM Interpreter. FSM interpreter retrieves the first message from the common message box and activates the corresponding FSM instance to make transitions according to the transition table and the received message.
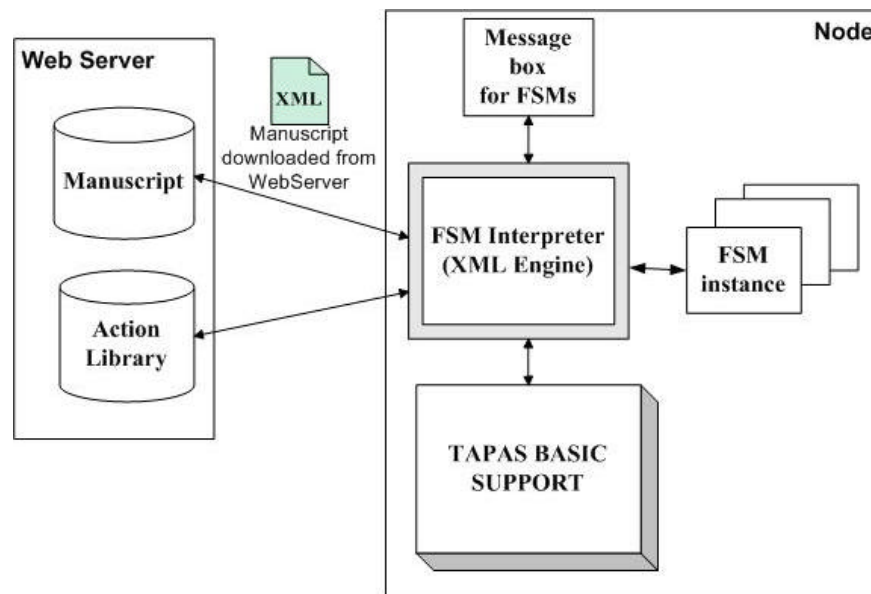
Figure 6. Implementation framework (engineering model)

In XML file, only method name and parameters for each action are specified in an <action> list (cf. Figure 5). The actual actions are implemented in Action Library depending on the underlying platform and language. They are also utilising node inherent capabilities, like printing and file server functions. The application specific data are recorded in FSM instances and real parameter values are provided by FSM instances during runtime. By utilizing Java Reflection [13], each time the FSM interpreter interprets the internal actions, it will invoke the implementation method in Action Library according to the method name and parameters specified in the manuscript. The description and implementation of actions can therefore be separated. Manuscripts and Action Library are created by application designer and are available from the Web server.

Actions include general-purpose actions, such as, printing, encryption, WindowsNew, WindowsClose, and UserLogOn; and application specific actions, such as, setFsmData. Engineering ontology of actions is, therefore, important for implementation of Action Library. However, instead of defining a new ontology for every service system, a standard and predefined one can be shared and reused if it is available. For example, the process ontology defined in DAML-S[15] is a good candidate. Action library can then be implemented in Java or any other language according to the ontology.

Figure 7 gives an example, which illustrates the structure of the extended TAPAS platform. XML Manuscripts, Action Library, and TAPAS Support System are available from Web-server. 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). FSM interpreter manages the Actors instantiated as FSM instances on each node. Director and the Actors which the director manages can reside on different nodes.
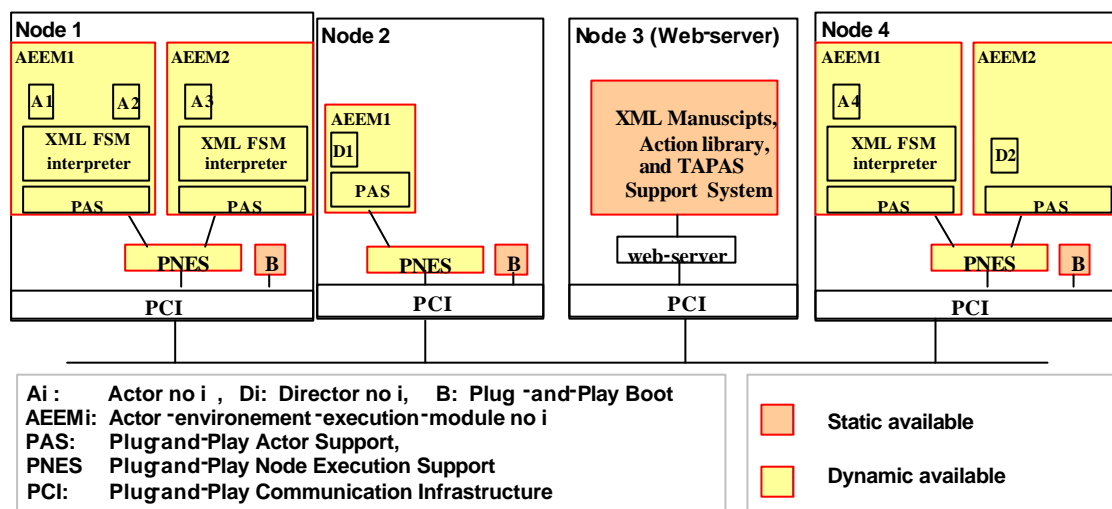

Figure 7. Example view of extended TAPAS platform for software execution

5. Conclusion

Modelling of behaviour is important for dynamic adaptable service development and deployment. An implementation language independent framework for behaviour description in XML is presented in this paper. This model is based on an FSM interpreter (XML engine), which is the interface between XML-represented behaviour description and runtime environment. FSM interpreter can be implemented in different languages and platforms, therefore, the model presented in this paper provides a solution for different systems to interoperate with each other and interchange data and behaviour. By defining or applying action ontology, an Action Library can be built for underlying platform, which utilises the node inherent capabilities and makes the XML-based behaviour description platform independent. The model has been implemented and integrated into TAPAS platform. Conversion between XML file and XMI can be provided as part of the functionality of the FSM Interpreter to make use of the benefits of XMI.

The result for the framework presented in this paper is a common XML representation for i) Dynamic service configuration and ii) Dynamic service instantiation. The XML-based

behaviour representation presented here will also be used in mobility handling architecture for actor movement. When an Actor is going to move from one node to another, all the FSM instance data will be packed in an XML file using the same structure as Figure 5, and sent to destination, where it will be unpacked by FSM interpreter so that the Actor can resume execution in the new node.

Service behaviour is not only procedures. For further perspective, the XML description makes it possible to reason behaviours based on some XML reasoning engine, such as an XET reasoning engine [16]. As an application for this, consider the interoperation between service systems based on TAPAS and other architectures. When a service request is given, there must be a dynamic mechanism for service discovery to match the required and provided services. The framework presented here can be used to reason between the service requirements and behaviours provided by the system based on the XML behaviour representation during service discovery.

References

1. Aagesen, F. A., Helvik, B., Johansen, U. and Meling, H. (2001) Plug and Play for Telecommunication Functionality: Architecture and Demonstration Issues. *Int'l Conf. Information Technology for the New Millennium* (*IConIT*), Thammasat University, Bangkok, Thailand.
2. Aagesen, F. A., Helvik, B., Anutariya, C., and Shiaa, M. M.(2003) On Adaptable Networking *ITC'03*. April 2003, Bangkok, Thailand.
3. Aagesen, F. A., Helvik, B., Wuvongse, V., Meling, H., Bræk, R. and Johansen, U. (1999) Towards a Plug and Play Architecture for Telecommunications. *Proc. 5th IFIP Conf. Intelligence in Networks* (*SmartNet'99*), Bangkok, Thailand. Kluwer Academic Publisher.
4. Aagesen, F. A., Anutariya, C., Shiaa, M. M. and Helvik, B. E. (2002) Support Specification and Selection in TAPAS. *Proc. IFIP WG6.7 Workshop and EUNICE Summer School on Adaptable Networks and Teleservices*, Trondheim, Norway, September, 109-116.
5. Shiaa, M., M. Mobility Support Framework in Adaptable Service Architecture. Network Control and Engineering for QoS, Security and Mobility 2003 IFIP/IEEE Conference (NetCon'2003), Muscat-Oman, October 2003.
6. W3C: eXtensible Markup Language (XML): http://www.w3.org/XML/.[June 12, 2003]
7. Bræk, R., Husa, K., E., Melby, G. ServiceFrame Whitepaper, *Ericsson NorARC*, May 2002.
8. SuperX++. http://xplusplus.sourceforge.net/index.htm [June 12, 2003]
9. Florescu, D., Grünhagen, A., Kossmann, D. *XL: An XML Programming Language for Web Service Specification and Composition* .WWW2002, International World Wide Web Conference, Honolulu, HI, USA, May 7-11, 2002.
10. Object Management Group. XML Metadata Interchange, version 1.1. ftp://ftp.omg.org/pub/docs/ad/99-10-02.pdf.
11. Nguyen, Hai Thanh. XML Based Data Exchange: Requirements and evaluation of XMI, GML and ISO 19118. Cand Scient Thesis. University of Oslo, 2001.

12. IBM developerWorks. Convert designs and code between Java, UML and Rational Rose. June 1, 2001. http://www-106.ibm.com/developerworks/webservices/library/co-cow21.html [June 12, 2003]

13. Sun Microsystems, inc. Java Core Reflection. http://java.sun.com/j2se/1.3/docs/guide/reflection/spec/java-reflectionTOC.doc.html [June 12, 2003]

14. Wuwongse, V., Anutariya, C., Akama, K. And Nantajeewarawat, E. (2001) XML Declarative Description (XDD): A language for the Semantic Web. *IEEE Intelligent Systems* 16(3):54-65.

15. The DAML Services Coalition. DAML-S: Semantic Markup for Web Services. http://www.daml.org/services/daml-s/0.7/daml-s.pdf. [June 12, 2003]

16. Anutariya, C., Wuwongse, V. And Wattanapailin, V. (2002) An Equivalent-Transformation-Based XML Rule Language. *Proc. Int'l Workshop Rule Markup Languages for Business Rules in the Semantic Web*, Sardinia, Italy.