An XML-based Framework for Dynamic Service Management

Mazen Malek Shiaa, Shanshan Jiang, Paramai Supadulchai and Joan J. Vila-Armenegol

Department of Telematics
Norwegian University of Science and Technology (NTNU)
N-7491 Trondheim, Norway
{malek, ssjiang, paramai}@item.ntnu.no , vilaarme@stud.ntnu.no

Abstract. Service systems are likely to be highly dynamic in terms of changing resources and configurations. On the one hand, resources are increasingly configurable, extendable, and replaceable. On the other hand, their availability is also varying. For this reason, the handling of these changes is crucial to achieve efficiency. To accomplish this objective, a framework for dynamic service management with respect to service specification and adaptation is proposed.

1 Introduction

A service system, in general, is viewed as a composition of service components. In the lifecycle of service systems (or service components) there are two main phases: the service specification and the service execution phases. The first handles the way services being specified, while the second comprises all the tasks related to service instantiation, operation and maintenance. Historically, service management as a concept has always been discussed and disputed within the second phase only, i.e. independently from the *Service Specification.* Manual modification of the service specification and thereafter configuration and reconfiguration are therefore needed. The concept of *Dynamic Service Management* will take a different approach to service management. It will propose procedures that will make services adaptable to the dynamic changes in their execution environment, based on modifiable and selectable service specifications.

In this paper, we propose a framework for Dynamic Service Management, which addresses two key issues; *Service Specification* and *Service Adaptation*. The main idea is to use behavior specifications with generalized action types as the service specifications. The actual executing code, or the action libraries that include routines specific to the execution environment, is determined during run-time. The system resources are represented by the so-called *Capability* and *Status* of the system, which characterize all the information related to resources, functions and data inherent to a particular node and may be used by a service component to achieve its functionality. *Service Adaptation* is achieved by allowing the service components to dynamically modify their functionality by requesting changes to their service specification. The framework uses web services to manage the availability and communication of service components.

2 Related work

Service management, and dynamic service management, has been dealt with in a number of papers, e.g. in active networks [1], replacement of software modules in [2]. In general, the principles discussed in [3] are considered to serve as a basis for any dynamic change management, which are also followed in this paper. Another approach, sharing our view of *Capability* availability, can be found in [4]. It addresses the problem of providing information access to mobile computers using the principle of adjusting the data according to the environment status. Each application is responsible for deciding how to exploit available resources by selecting the fidelity level of the transmitted data. The adaptation is based on choosing between different versions of the data (fidelity levels) in order to match the resource availability. Our work targets the adaptation of any kind of behavior, instead of the adaptation of data itself. It requires that the behavior is rich in its processing possibilities. In this regard, the support platform (that executes in the nodes) is responsible for monitoring resource availability, notifying applications of relevant changes to those resources and enforcing resources allocation decisions. Each application is responsible for deciding how best to exploit available resources.

3 Service Specification

As a basic assumption in the framework, a service is viewed as a composition of one or more service components (we also consider a service as a *play* consisting of different *roles*.) Each Service Component is realized or carried out by one or more Role-Figure (being an entity in the architecture that is capable of representing some well-defined functionality). A Role-Figure is realized by a software component (or a collaboration of software components, e.g. multi-threaded processes) executing in a node. However, throughout this paper, and as an abstraction from the implementation domain, Role-Figure will be the constituent of the architecture that is used to provide a basis for service specification and instantiation. In this regard, a Role-Figure specification is the service (or part of a service) specification, which gives a service behavior description. The most intuitive way to model such a specification is by a State Machine model (one well-known and applied model is the Extended Finite State Machine, EFSM, that is considered here.) [5]. Figure 1 shows the EFSM data structure for the Role-Figure specification. The behavior description defined in the Role-Figure specification consists of *states*, *data* and *variables*, *inputs*, *outputs*, and different *actions*. Actions are functions and tasks performed by the Role-Figure during a specific state. They may include calculations on local data, method calls, time measurements, etc. The <*Action*> list in the Role-Figure specifications specifies only the action type, i.e. the method name, and parameters for the action. Each action type belongs to some Action Group according to the nature of action.

Declaring Actions by their general types and classifying them into Actions Groups (e.g. *G1*: Node Computation Capability, *G2*: Communication model, *G3*: Graphics, *G4*: I/O interaction) is a technique used to tackle the problem of platform and implementation independence, as well as achieving a better flexibility and reusability in service and application design. Using Action Types and Action Groups in the service specification keeps the specification short, clean and abstract from how it would eventually be implemented in different end-user devices, operating policies, and executing environments. For instance, actions such as terminate, exit, error handling, etc. can be classified in "*G100: Control Functions*". Consequently, the action *terminate* would be used in a specification as: <actionType>*terminate*</actionType> <actionGroup>*G100*</actionGroup>

The *Capability* concept abstracts all the information related to functions, resources and data required by the Role-Figures to achieve their functionality. Examples of such capabilities can be: software/hardware components, such as units of processing, storage, communication, system data, etc. Role-Figures achieve tasks by performing or executing actions. Such actions would naturally consume or require resources, or capabilities. A Role-Figure specification explicitly specifies what sorts of capabilities are required. Occasionally, the execution of the Role-Figure halts if certain capabilities are not available. *Status* comprises observable counting measures to reflect the resulting state of the system.

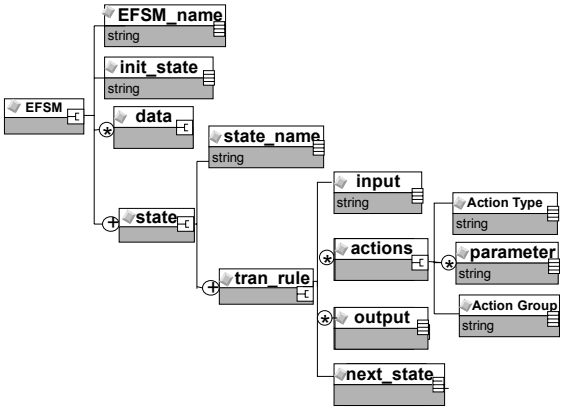

Figure 1 EFSM model data structure.

As have been indicated previously, Action Types, and eventually their classes or Action Groups, are not "*executables"* that may be run in a specific environment or device. Therefore, service designers should map their actions to executable routines provided by device manufacturers (Usually using built-in function calls, with explicit and proper parameter values, through the device's Application Programming Interface, or API.) In this regard, these executable routines should also be classified, to indicate what operating circumstances and capability requirements, they are working within and demanding for. We refer to this classification as *Capability Categories,* so that each category represents a capability set. Examples of Capability Categories can be: *C1*: Powerful PDA, *C2*: Basic PDA, *C10*: Bluetooth, *C11*: WLAN, *C100*: Default CC for G100. A mapping is therefore required to link the action definitions in the service specification to the executable routines stored in Action Libraries.

4 Dynamic Service Management Framework

The Framework presented here considers three distinct forms of Role-Figure specification. Firstly, Role-Figure specification exists as a static representation of the behavior of the Role-Figure functionality. Secondly, Role-Figure specification would exist as an instantiated code or class instances, with all the necessary mappings to their executing environment, which is "*instantiated Role-Figure specification*." However, a third form may exist between these two forms. Once the capability category is determined, it is important to extend and convert certain actions into

corresponding sets of actions, e.g. providing extra security and authentication checks. This form we refer to as "*calculated Role-Figure specification*."

The framework for Dynamic Service Management is illustrated in Figure 2. The components of the framework are:

1. *Play Repository:* a data base that contains the service definitions and includes:
   - *Role-Figure Specifications*: provide behavior definitions for each Role-Figure, including the Action Types to be performed and their corresponding Action Groups.
   - *Selection Rules*: provide information for dynamically selecting the proper Role-Figure Specification, if it has several corresponding specifications.
   - *Mapping Rules:* specify the mapping between capabilities and capability categories.
2. *Capability and Status Repository (CSRep):* is a database that provides a snapshot of the resources of the system. It maintains information on all capability and status data in all system nodes.
3. *Action Library:* is a database that contains codes for the state machine-based actions. These codes are implemented according to the capability category they require.
4. *Service Manager (SM):* is responsible for the handling of *Initial Service requests* (to instantiate a Role-Figure), *Role-Figure move* (to move an already instantiated Role-Figure from one node to another), and *Function Update requests* (to change the functionality of an already instantiated Role-Figure). It, first, selects a specific Role-Figure specification based on the *Selection Rules.* Secondly, it calculates the offered Capability Category according to the *Mapping Rules*, which is denoted as *Mapping table.* Then it generates the *calculated Role Figure Specification* by adding the corresponding Capability Category information and the substructures that can be used for decision-making when capability changes occur. This *calculated Role Figure Specification* and the *Mapping table* are then sent to the proper *State Machine Interpreter* for execution.
5. *Requests:* supply, on the one hand, the identification of the Role-Figure to be instantiated or modified. On the other hand, they provide the information to be taken into consideration during the calculation of the Capability Category. Three types of service requests may be handled by the SM:
   - *Initial Service request* indicates a role to be executed in a node.
   - *Role-Figure move* is issued when there is a severe deterioration in certain capabilities availability or the Role-Figure is requested to move to achieve a mobility task for instance. [6] gives an overview of the mobility management of Role-Figures.
   - *Function Update request* is issued to update a functionality due to capability change.
6. *Results:* are the outcomes of the calculations performed by SM, which contains the following:
   - *calculated Role-Figure Specification* indicates a changed Role-Figure specification.
   - *Mapping table* is the result of calculating and matching of Capability Categories based on the given instantaneous capability situation, Mapping rules, and incoming request and its parameters.
7. *State Machine Interpreter (SMI*): is a State Machine execution support [5]. This is the primary entity in the framework responsible for the execution of Role-Figures according to the *instantiated Role-Figure Specification*. The framework allows for a decentralized computation. The dotted-arrow connecting the *Play Repository* and the SMI allows for the calculation of the Role-Figure Specification to be conducted in the node where it will execute, i.e. by the SMI instead of the SM that is, in most cases, would exist at a remote location. This option can solve problems related to over-loaded SM, congested network, time-critical applications, etc.

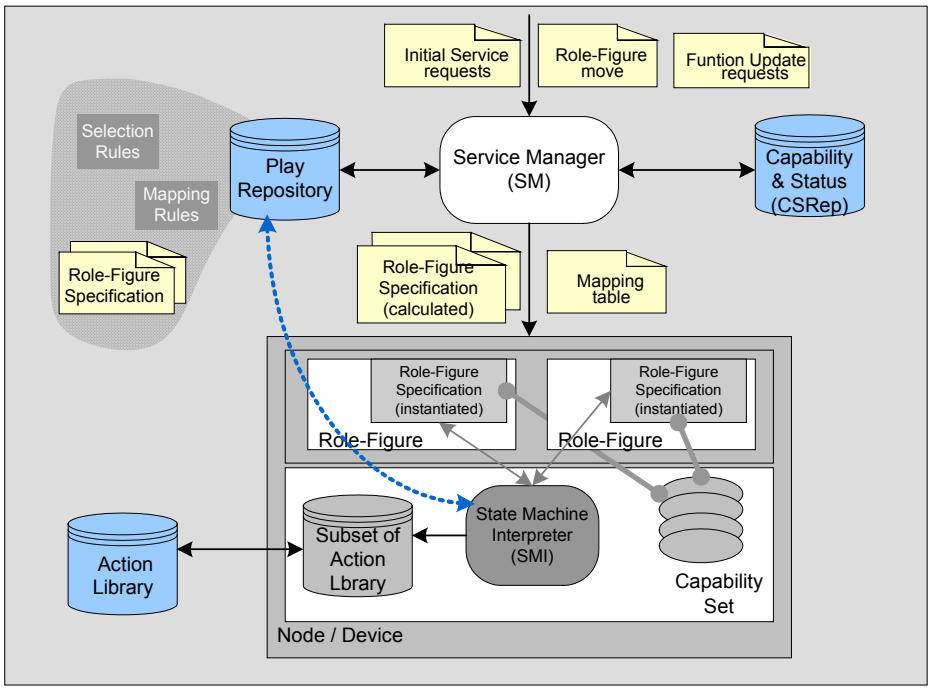

Figure 2 Dynamic Service Management Framework.

The next specification is of a *clientMultimediaPlayer* Role-Figure that runs on a Laptop, with both the capability of WLAN (default) and Bluetooth (used when WLAN is unavailable).

```
<state name="stMediaPlay">
  <actionType>MediaPlay</actionType>
  <actionGroup>G2</actionGroup>
  <CapCategory>C10</CapCategory>
  <CapCategory>C11</CapCategory>
  <Config>
      <defaultCC>C10</defaultCC>
      <ProblemType>
         <List>out of coverage</List>
         <List>congestion</List>
      </ProblemType>
      <choice>
         <check>Check Bluetooth neighbourhood</check>
         <substate name="Bluetooth">
            <condition value="available">
            <output>
               <msg type="FunctionUpdate">
                  <param>
                     <name>manuscript</name>
                     <value>SchoolClient</value>
                  </param>
                  <param>
                     <name>C10</name>
                     <value>out of coverage</value>
                  </param>
                  <param>
                     <name>Wireless Communication</name>
                     <value>unavailable</value>
                  </param>
                  <dest>ServiceManager</dest>
               </msg>
            </output>
            <nextState>stWaitForManuscript</nextState>
         </substate>
         <substate name="NoBluetooth">
            <condition value="unavailable" offline="No">
            <actionType>Terminate</actionType>
            <actionGroup>G100</actionGroup>
            <CapCategory>C100</CapCategory>
            <nextState>stTerminate</nextState>
         </substate>
         <substate name="Offline">
            <condition value="unavailable" offline="Yes">
            <actionType>ChangeToOffline</actionType>
            <actionGroup>G100</actionGroup>
            <CapCategory>C100</CapCategory>
            <nextState>stWaitUserInput</nextState>
         </substate>
      </choice>
  </Config>
  <nextState>stWaitUserInput</nextState>
</state>
```

Part of a calculated specification for a *clientMediaPlayer* Role-Figure, in which an action of simple communication has been converted into a structure of additional actions and substates.

5 Implementation issues

The framework has been developed and implemented as part of the TAPAS architecture, see [6,7,8] and the URL: http://tapas.item.ntnu.no/. The implementation of the framework is built around the support functionality of the TAPAS core platform. Java Web Services Developer Pack (Java WSDP) [9] was applied to develop the main communication parts of the framework, while Apache Axis [10] was used as a SOAP server. In this regard, nodes running the platform will have an entity that supports Web Services requests and replies. Figure 3 shows a possible implementation of the framework, in which a configuration of two nodes running two and three distinct Role-Figures is applied, beside a node running the SM and a web server containing the repository data. The TAPAS Core Platform has been extended with Web-services communication routines, node registry capabilities, and extended configuration data reflecting the reachability of SM. In the figure a connection is highlighted between the CSRep and the nodes participating in the execution of these Role-Figures. Although it has not been fully implemented, a capability registration and update mechanism is considered, which keeps the CSRep updated in terms of any capability change in the nodes. Throughout the experimentation process such update has been conducted manually in order to simplify the overall processing.

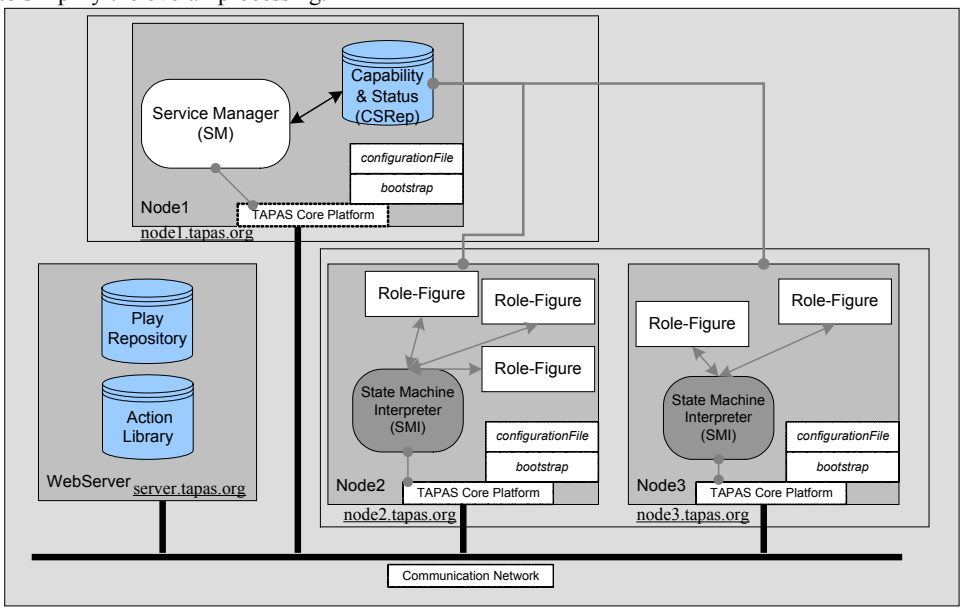

Figure 3 An implementation of the Dynamic Service Framework within the TAPAS platform

6 Experimentation Scenarios

Several scenarios have been proposed to demonstrate the applicability and foremost features of the framework. During the experimentation, simple application scenarios have been used. The application used was the Teleschool, which is an application facilitating distance-learning, allowing students and teachers to participate in virtual class activities in real-time or off-line modes, using multimedia capabilities on various types of terminals and devices. The test scenarios were limited to run and execute the client Role-Figures, and examine their proper functionality. Here we instantiate *SchoolClient* Role-Figure on a node featuring a Bluetooth and WLAN communications. Below we show an example of a *Role-Figure move* request to initiate a move functionality of a Role-Figure from a node to another one.

```
<RoleFigureMoveRequest>
 <sender><oNode>http://Node2.tapas.org</oNode>
  </sender>
  <dateTime />
  <serviceType>Teleschool</serviceType>
  <roleRequesting>SchoolClient</roleRequesting>
  <RFSUsed>SchoolClient_Advanced</RFSUsed>
  <preferredConfiguration>
      <nodeInstalling>
          http://Node3.tapas.org
       </nodeInstalling>
  </preferredConfiguration>
  <contextInfo>
    <connectionUsed>LAN</connectionUsed>
    <userSubscription>Advanced</userSubscription>
    <MMSupport>VideoPlayer</MMSupport>
  </contextInfo>
  <stateInfo>
      <currentState>stInitUserInterface
      </currentState>
      <variables>
          <variable>
              <name>v_server</name>
              <value>aaaaaa</value>
          </variable>
      </variables>
  </stateInfo>
</RoleFigureMoveRequest>
```

An example *Role-Figure move* request used to move a Role-Figure from Node2 to Node3

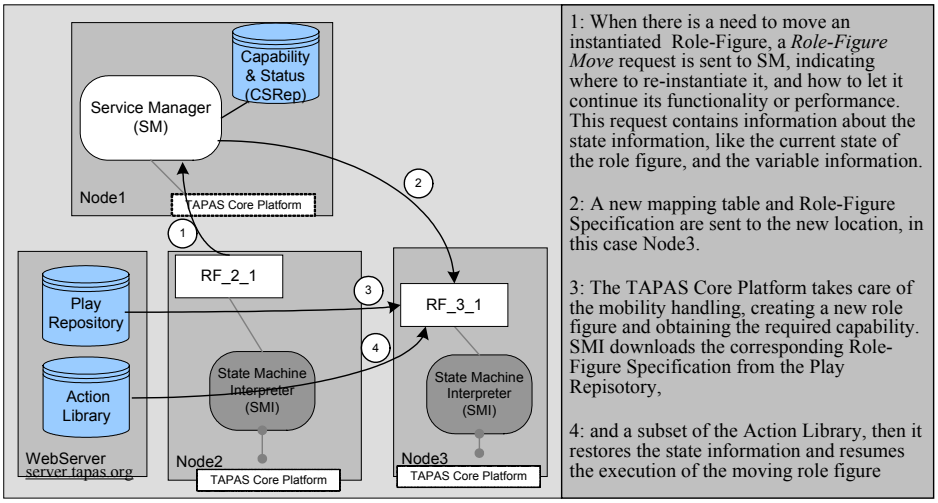

Figure 4 Experimentation scenario of a *Role-Figure move* including two nodes.

7 Conclusion

In this paper some challenges of the Dynamic Service Management have been discussed, and a framework has been proposed to tackle them. In a highly dynamic system, components composing the system come and go, as well as the system resources that are allocated for them vary and change all the time. The demonstrated framework provides a way of enabling the dynamic service management based on specifications that can be selected, their behavior be computed, and their handling of system resources are all based on the available capabilities in the execution environment. Specifications contain only Action Types and Action Groups, while the executable code or Action Libraries are available on a different database. This distinction is mainly to achieve flexibility. Capability Categories classify these executables based on the capability information in the system.

References

1. M. Brunner and R. Stadler, Service Management in Multiparty Active Networks, *IEEE Communications Magazine*, Vol. 38, No. 3, pp. 144-151, March 2000.
2. Christine R. Hofmeister and James M.Purtilo. Dynamic reconfiguration in distributed systems: Adapting software modules for replacement. *In Proceedings of the 13th International Conference on Distributed Computing Systems, IEEE Computer Society Press,* May 1993.
3. Jeff Kramer and Jeff Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering,* 16(11):1293-1306, November 1990.
4. B.D. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in Odyssey (1999). *Mobile Networks and Applications*, 4, 1999.
5. Jiang, S. and Aagesen, F. A. (2003) XML-based Dynamic Service Behaviour Representation. *NIK'2003*. Oslo, Norway, November 2003. [http://tapas.item.ntnu.no]
6. Shiaa M. M and Aagesen. F. A. (2002) Architectural Considerations for Personal Mobility in the Wireless Internet, *Proc. IFIP TC/6 Personal Wireless Communications (PWC'2002)*, Singapore, Kluwer Academic Publishers, October 2002. [http://tapas.item.ntnu.no]
7. Aagesen, F. A., Anutariya, C., Shiaa, M. M. and Helvik, B. E. (2002) Support Specification and Selection in TAPAS. *Proc. IFIP WG6.7 Workshop on Adaptable Networks and Teleservices*, Trondheim Norway, September 2002. [http://tapas.item.ntnu.no]
8. Aagesen, F. A., Helvik, B. E., Anutariya, C., and Shiaa M. M. (2003) On Adaptable Networking, *Proc. 2003 Int'l Conf. Information and Communication Technologies (ICT 2003)*, Thailand.
9. SUN Microsystems, the Web services Homepage, Java Web Services Developer Pack (WSDP) documentation, http://java.sun.com/webservices/index.jsp
10. The Apache homepage, Apache Axis 1_1, http://ws.apache.org/axis/