Configuration Management for an Adaptable Service System

Finn Arve Aagesen* — Paramai Supadulchai* — Chutiporn Anutariya — Malek Mazen Shiaa***

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

aagesen@item.ntnu.no
paramai@item.ntnu.no
malek@item.ntnu.no

**Computer Science Program, Shinawatra University*
Pathumthani 12160, Thailand

chutiporn@shinawatra.ac.th

ABSTRACT: *Adaptable service systems are service systems that are capable of handling dynamic changes in both time and position related to users, capabilities, nodes and changed service requirements. The paper presents a formal framework for dynamic configuration and reconfiguration of services in TAPAS (Telematics Architecture for Play-based Adaptable Systems). The framework presented in this paper, provides representation and reasoning mechanisms for semantic description and matching of required and offered capabilities and status which are required by a particular service system. It employs CIM and RDF based on XML as well as the XML Declarative Description Language (XDD) to provide human-readable and machine-comprehensible descriptions of status, capabilities, system (re)configuration plans as well as the exchange of messages. A reasoning system for Configuration Management has been developed by use of XET (XML Equivalent Transform).*

This system can directly operate and reason about XML elements and XML clauses described by XDD. The system is demonstrated for a simple Intelligent Printing Management System.

KEY WORDS: *Autonomic Communication, Adaptable Systems, Dynamic configuration, Configuration management.*

1. Introduction

A network based service system consisting of services, service components and nodes is considered. A service is realised by the structural and behaviour arrangement of service components, which by their inter working provide a service in the role of a service provider to a service user. Service components are executed as software components in nodes, which are physical processing units such as servers, routers, switches and user terminals. User terminals can be phones, laptops, PCs and PDAs etc.

Network-based services have during more than one decade been an important research topic. Example topics include TINA (Tele-communication Information Networking Architecture) (Inoue at al. 1999), Mobile Agents and Active and Programmable Networks (Bieszczad et al. 1998) (Raza et al. 2004) (Tennenhouse et al. 1997). 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. Traditionally, the nodes as well as the service components have a predefined functionality. Concerning nodes as well as software engineering principles, changes are taking place. Nodes are getting more generic. A modern node may offer IP telephony and can have an MP3 player, video camera, storage etc. In the same way, the software components are getting more generic. From being static components, the software components can be generic software components, which are able to download and execute different functionality depending on the need. Such generic programs are from now on denoted as *actors*. The name actor is chosen because of the analogy with the actor in the theatre, which is able to play different roles in different plays.

We are entering a generative era, which gives a high degree of flexibility. To utilise the generative potential, the attributes of services, service components, software components and nodes must be appropriately formalised, stored and made available. There must also be generative platform functionality that utilises this generative data. Generative data and functionality apply to the ordinary service functionality, but also to the service management functionality. As a first step towards this formalisation, the concepts *capability* and *status* are introduced.

A capability is an inherent property of a node or a user, which defines the ability to do something. A capability in a node is a feature available to implement services. An actor executes a program. However, this program may need capabilities in the node. A capability of a user is the feature that makes the user capable of using services. Capabilities can be classified into:

– Resources: physical hardware components with finite capacity,

– Functions: pure software or combined software/hardware components, which perform particular tasks,

– Data: just data, which interpretation, validity and life span of which depend on the context of the usage.

Resource capability examples are processing, storage and communication resources e.g., CPU, hard disk and transmission channels, standard equipment e.g., printers and media handling devices and special equipment e.g., encryption devices. Function capabilities are functions related to the use of hardware resources, such as encryption, and special programs or library functions available for general use. Data capability examples are user login and access rights

Status is a measure for the situation in a system with respect to the number of active entities, the traffic situation and the Quality of Services (QoS) etc. Status reflects an instantaneous state of the system. It can comprise observable counting measures, or calculated QoS measures.

The work presented in this paper has been related to the *Telematics Architecture for Play-based Adaptable System* (TAPAS) (Aagesen et al. 1999, 2001, 2002, 2003). The TAPAS *computing architecture*, to be more detailed explained in Section 3, defines a service system by a play. A play consists of several actors, constituting role figures by playing roles. A role figure is realised in an executing environment in a node, and is utilising capabilities, which are inherent properties of the node. A role can have specific requirements to capabilities and status. Due to the dynamic availability of nodes in the network as well as changes in their capabilities and status, it is desirable that configuration of services is done dynamically. *Configuration management* is the optimisation of service systems initial configuration and reconfigurations with respect to capabilities and status. This is the focus of this paper.

Section 2 discusses related works. Section 3 gives a brief outline of TAPAS architecture. Section 4 proposes a dynamic configuration framework. Its data model and reasoning mechanism are presented in Section 5. Section 6 demonstrates a practical application of the framework together with the reasoning mechanism. Section 7 concludes and outlines further research direction.

2. Related work

Several configuration management and adaptable architectures have been proposed so far (Bakour et al. 2004) (Cohen et al. 2004) (D'Antonio et al. 2004) (Keller et al. 2004) (Sahai et al. 2004) (Solarski et al. 2004). Nevertheless, they are most likely the architectures to handle a specific task, which either can be the *service creation and deployment functionality* (Bakour et al. 2004) (Cohen et al. 2004) (Keller et al. 2004) (Solarski et al. 2004) or the *network and resource management functionality* (D'Antonio et al. 2004) (Sahai et al. 2004). Our architecture is intended to provide a configuration management for any adaptable system that provides the functionality for both service creation and deployment network and resource management. This diversity comes from the use of *XML Declarative Description (XDD)*, a generic knowledge representation. XDD provides a single uniform formalism to create knowledge that incorporates various capability and status representations as well as service behavior representations. Moreover, the ability to effectively handle different kinds of event messages, which are well categorised in an ontology instance, and the underlying reasoning mechanism guarantee that an event happening in the system will be handled by rule-based procedures that can apply to them. The reasoning mechanism transforms an event message *equivalently* with the supplied configuration rules until a proper procedure to handle the event is obtained. The transformation preserves all the semantic in a service system (Wuwongse et al. 2001).

3. TAPAS architecture

TAPAS intends to be an architecture that gives 1) rearrangement flexibility, 2) failure robustness and survivability, and 3) QoS awareness and resource control (Aagesen et al. 2003). The TAPAS architecture is separated into a *computing* architecture and a *system management* architecture as follows:

– The computing architecture is a generic architecture for the modeling of any service software components

– The system management architecture is the structure of services and service management components.

These architectures are not independent and can, to some extent, also be seen as architectures at different abstraction layers. The system management architecture, however, has focus on the functionality independent of implementation, and the computing architecture has focus on the modeling of functionality with respect to implementation, but independent of the nature of the functionality. The nature of the computing as well as system architecture is described briefly in the following.

3.1. Computing architecture

TAPAS computing architecture has three layers: the *service view*, the *play view* and the *network view* as shown in Figure 1. The service view concepts are rather generic and should be consistent with any service architecture. Likewise, the network view concepts are generic and should be consistent with any corresponding network architecture, with exception of the *core platform*, which is a specific platform supporting the play view concepts. The network view concepts are the basis for implementing the play view concepts, which again are the basis for implementing the service view concepts. In the other way around, the service view concepts are mapped into the play view concepts, which again are mapped into the network view concepts.

The play view intends to be a basis for designing functionality that can meet the requirements related to rearrangement flexibility, the failure robustness and survivability, and the QoS awareness and resource control. The play view concepts are seemingly rearrangement flexibility oriented. The capability and status concepts, however, also give a basis for the further design of systems that can meet the failure robustness and survivability as well as the QoS awareness and resource control requirements.

In the network view, *nodes* are typically network processing units such as mobile phone, desktop computer, laptop, printer and router that possess particular capabilities. Nodes are installed with *core platform*. Core platform supports basic communication infrastructure between nodes. At a specific time point, status is the state of a system with respect to the number of active entities, traffic situation and QoS etc.

The play view concepts are founded on a *theater metaphor*. The TAPAS actor is a generic software component consistent with the actor definition given in Section 1. However, the TAPAS actor is specialised as follows. Actors perform *roles* according to predefined *manuscripts*, and a *director* manages their performance. Actors are software components in the nodes that can download manuscripts. An actor will constitute a *role figure* by behaving according to a manuscript that defines the functional behavior of that particular role in a *play*. A *role session* is a projection of the behavior of a role figure with respect to one of its interacting role figures. Actors in TAPAS can be moved transparently between nodes and the role sessions between them can be re-instantiated automatically (Shiaa 2004).

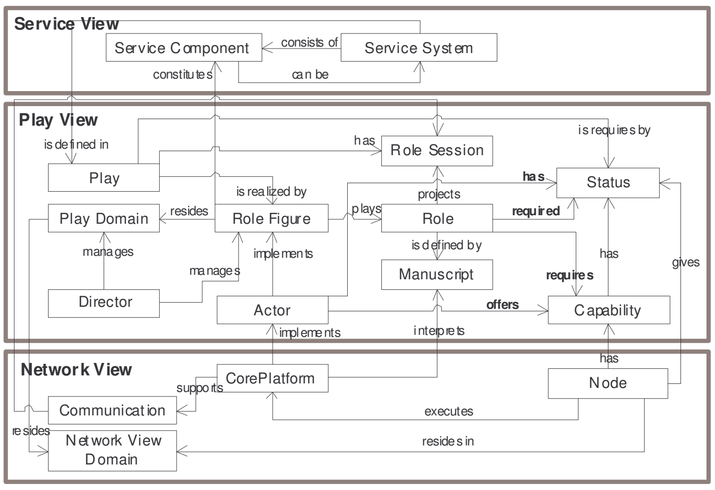


Figure 1. The Simplified TAPAS Computing Architecture

A director is an actor with supervisory status regarding other actors. A director also represents a play view domain, which is a set of nodes, which actors are supervised by a single director. The director chooses a fitting actor for a certain role figure. For this task the director requests help from the service management functionality defined in Section 3.2.

A *service system* is defined by a *play*. A play consists of several actors playing different roles, each possibly having different requirements on capabilities and status. An actor will constitute a role figure, based on the role defined by a manuscript. The ability of an actor to play a role depends on the matching of the required capabilities and status of the role and the offered capabilities and status in the node where of the actor is executing. TAPAS Core Platform supports the play view concepts (Aagesen et al. 2003).

3.2. System management architecture

The main functionality components of the system management architecture are illustrated in Figure 2. To fulfill the failure robustness and survivability requirements, the architecture must be dependable and distributed. This means that replication of resources and functionality is needed. The dependability aspect is beyond the scope of this paper, and the various functionality components will be defined as being part of a centralised architecture. The *Primary Service Providing Functionality* comprises the ordinary services offered to human users. In addition, the following functionality components are defined:

– *Service Management*: Definition of new services, deployment and invocation of services and service components

– *Capability and Status Management*: Registration, de-registration, update, transform and provide access to capabilities and status repository.

– *Configuration Management*: Optimisation of service systems initial configuration and re-configuration with respect to the capabilities and QoS.

– *Mobility Management*: The handling of various mobility types.

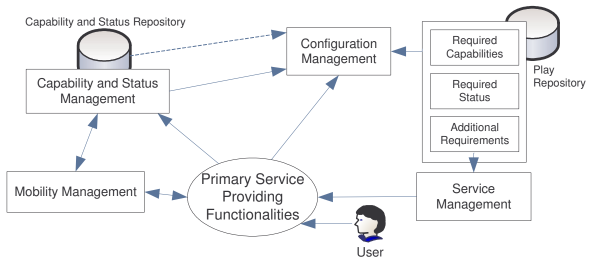


Figure 2. System management functionality components

The functionality of these functionality components is constituted by the cooperation of role figures. Each of these functionality components has one *dedicated main role figure*, acting as the visible interface to the other components. This main role figure is denoted as the manager. In this paper, a functionality component is considered to consist of the manager only. The functionality components defined above are accordingly replaced by the *Service Manager, the*

Capability and Status Manager, the Configuration Manager and the Mobility Manager, respectively. This paper has focus on Configuration Management and the Configuration Manager. Aspects of the other functionality components without relevance to Configuration Management are beyond the scope of this paper.

4. Dynamic configuration framework

Figure 3 describes an architectural framework for dynamic configuration and reconfiguration of services.

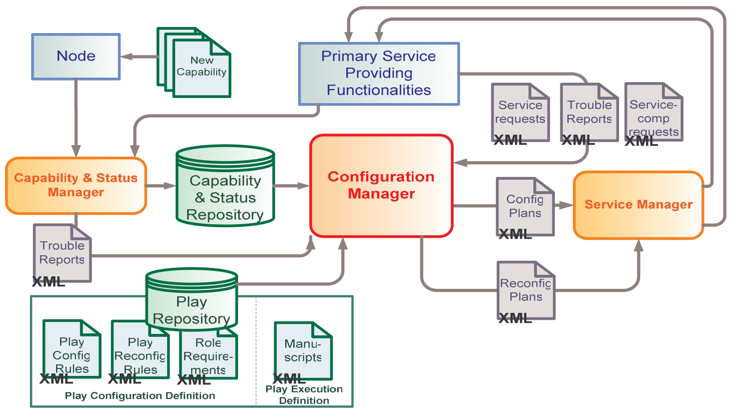


Figure 3. Architectural framework for dynamic configuration

The main entities are the Configuration Manager (CM), the Capability & Status Repository (*CSRep*), the Play Repository, the Capability and Status Manager, and the Service Manager.

The Configuration Manager is responsible for:

– Generation of appropriate configurations for composing new services to be installed in a system: When there arises a request for installing a new service (i.e., a service request), the CM fetches a corresponding play definition and retrieves the system capabilities and status from the Play Repository and the Capability and Status Repository, respectively. Valid configurations for such a service are generated and analysed, and an appropriate configuration will be selected based on the specified selection criteria such as system performance and QoS, user preferences and cost. The selected configuration (ConfigPlans), defining which nodes in the system should execute actors constituting certain roles, will be forwarded to and executed by the Service Manager.

– Determination of a location for executing a particular role: In the running 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 Manager to load a corresponding manuscript from the Play Repository and instantiate it on the suggested node.

– Determination of reconfiguration schemes for dynamic reconfiguration of existing service systems: Upon the receipt of a trouble report indicating a problem in a running system, the CM analyses the problem, fetches related information from the Capability and Status Repository and the Play Repository, and computes a service reconfiguration plan (ReconfigPlans) to be executed by the Service Manager. Possible plans include actor relocation, re-initialisation, load balance and distribution. Selection of an appropriate plan depends on the defined reconfiguration rules as well as the nature of a problem.

The *Capability and Status Manager* monitors system capabilities/status and maintains the Capability and Status Repository. It also 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. Capability and Status Manager is also responsible for installation and de-installation of capability components. When a new node with not-yet-installed capability components is plugged into the system, these components will be installed according to certain well-defined procedures, and their capabilities will be registered as part of the system. Similarly, de-installation of a component requires an execution of certain procedures and deregistration of the component's capabilities.

The *Service Manager* installs a service into the system by creating corresponding actors for execution of certain roles according to an obtained play configuration or reconfiguration plan generated by CM. Allocation of capabilities as well as instantiation of a manuscript for each role are also performed by this entity.

The *Capability & Status Repository* stores specifications of capabilities offered by components in a system and maintains information reflecting the situation and status of the system. Such status information can be certain environment conditions, observable values of the current QoS characteristics as well as their calculated measures, which will be analysed by the Configuration Manager when computing (re)configuration plans for the system.

The *Play Repository* is a collection of *play configuration definitions* and *play execution definitions*. A *play execution definition* consists of *manuscripts,* which define the entire functional behaviour of each role in terms of EFSM (Extended Finite State Machine). A *play configuration definition* is an aggregation of the three specifications:

– *Role requirements* identify, for each role, its requirements on available capabilities/status.

– *Play configuration rules* describe system configuration rules and 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, optional and mandatory constraints as well as conflict handling and priority information.

– *Play reconfiguration rules* define policies for the handling of reconfiguration related events, such as service component failure, decrease in system QoS and resource unavailability.

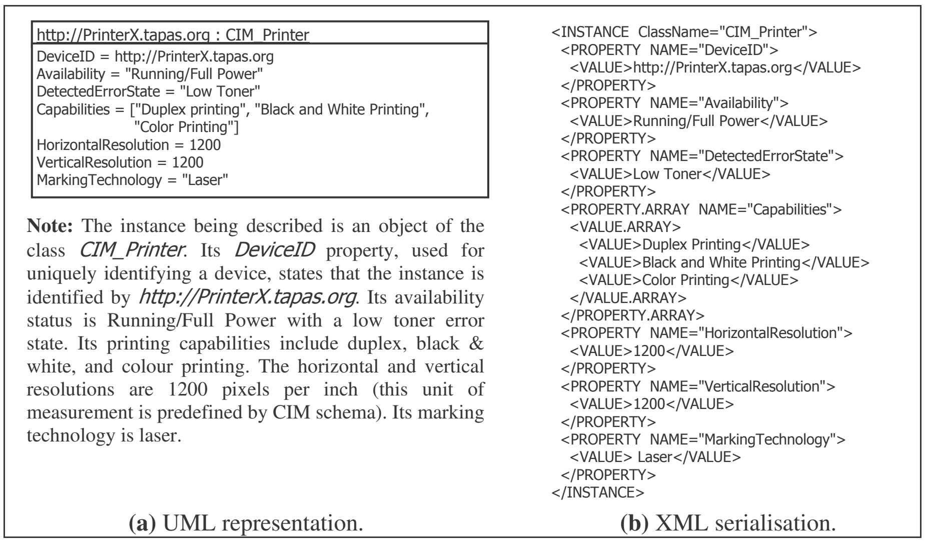
5. Data model

This section presents XML based approaches to the representation of a dynamic configuration data model. It elaborates machine-comprehensible descriptions for each component of the configuration framework presented in Section 4 by the application of standard languages for the Semantic Web (Berners-Lee et al. 1999) and network management. First, the description of status and capabilities is discussed, followed by the message specification modelling. Finally formalisms for play configurations and reconfiguration definitions are presented.

5.1. Capability and status specification

The developed framework proposes the use of standard XML-based metadata and ontology languages for modelling and providing semantic description of system capabilities and status. RDF (Brickley et al. 2004) (Lassila et al. 1999), which is a W3C recommended metadata language and its extensions (e.g., DAML (Hendler et al. 2000) and OWL (McGuinness et al. 2004)) appear to meet this language need. However, so far a standard, common ontological schema for describing network management resources in such languages does not yet exist. Therefore, the framework employs and extends CIM schema (Westerinen et al. 2000), developed by DMTF (Distributed Management Task Force) for representing capabilities and status. CIM is a fundamental, yet comprehensive object-oriented schema, both with respect to classifications and associations of objects, for describing network management information in a standard MOF (Managed Object Format) and XML format. In CIM model, the notions of capabilities and status are represented together as parts of an object's properties. Figure 4 gives an example of a CIM instance, represented in both UML graphical notation and its XML serialisation; it describes capabilities, status and certain operational attributes of a printer.

Based on these modelling concepts, the Capability and Status Repository is then represented as a collection of CIM instances which describe the available capabilities and status of the plug-and-play system. Note that to conform to W3C standards, CIM schemas and instances encoded in RDF can also be used in the proposed framework. However, in the open, heterogeneous environment, it is impossible to assume that every component/system will solely employ CIM model for semantic description of its capabilities and status. Thus, with this concern, research on integration of different capability/status ontologies is also part of the TAPAS project



```
http://PrinterX.tapas.org : CIM_Printer
DeviceID = http://PrinterX.tapas.org
Availability = "Running/Full Power"
DetectedErrorState = "Low Toner"
Capabilities = ["Duplex printing", "Black and White Printing",
               "Color Printing"]
HorizontalResolution = 1200
VerticalResolution = 1200
MarkingTechnology = "Laser"
```

Note: The instance being described is an object of the class *CIM_Printer*. Its *DeviceID* property, used for uniquely identifying a device, states that the instance is identified by *http://PrinterX.tapas.org*. Its availability status is Running/Full Power with a low toner error state. Its printing capabilities include duplex, black & white, and colour printing. The horizontal and vertical resolutions are 1200 pixels per inch (this unit of measurement is predefined by CIM schema). Its marking technology is laser.

```
<INSTANCE  ClassName="CIM_Printer">
  <PROPERTY  NAME="DeviceID">
    <VALUE>http://PrinterX.tapas.org</VALUE>
  </PROPERTY>
  <PROPERTY  NAME="Availability">
    <VALUE>Running/Full Power</VALUE>
  </PROPERTY>
  <PROPERTY  NAME="DetectedErrorState">
    <VALUE>Low Toner</VALUE>
  </PROPERTY>
  <PROPERTY.ARRAY  NAME="Capabilities">
    <VALUE.ARRAY>
      <VALUE>Duplex Printing</VALUE>
      <VALUE>Black and White Printing</VALUE>
      <VALUE>Color Printing</VALUE>
    </VALUE.ARRAY>
  </PROPERTY.ARRAY>
  <PROPERTY  NAME="HorizontalResolution">
    <VALUE>1200</VALUE>
  </PROPERTY>
  <PROPERTY  NAME="VerticalResolution">
    <VALUE>1200</VALUE>
  </PROPERTY>
  <PROPERTY  NAME="MarkingTechnology">
    <VALUE> Laser</VALUE>
  </PROPERTY>
</INSTANCE>
```

(a) UML representation. **(b)** XML serialisation.

Figure 4. A CIM instance describing a printer device

5.2. Message specification

Because CIM does not provide means for representing various types of messages required by the developed architecture, RDF is exploited. Figure 5 illustrates various types of messages and gives their primitive attributes. Basically, each message carries its URI (Universal Resource Identifier), information of the actor who sends the message and the date/time of composing it. A sender's information also includes the installing location and the playing role. Other message attributes can also be encoded depending on the purpose of the message.

Messages are classified into two main types: *requests* and *trouble reports*. Requests are further divided into: *service request* and *service component request*. The former is a request for installation and execution of a particular service system which has not yet been installed while the latter is a request for instantiation of a particular service component in a running service system. Figure 6 gives examples of both types of requests.

Trouble reports are further classified into: *QoS degradation report* and *actor error report*, which are used for notifying the CM when a QoS-sensitive service system encounters a decrease in its QoS to an unsatisfactory level and when an actor-involving problem occurs, respectively. There are two types of actor error reports: (i) *Actor unreachable* is used when an actor in a running system wants to communicate and cooperate with another existing actor which constitutes a particular role but is somehow unreachable or not responding. (ii) *Insufficient capability* is sent by an actor to the Configuration Manager if the node where it is running has insufficient capabilities.

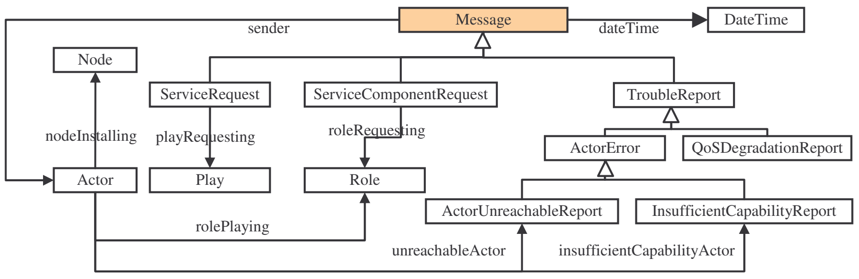


Figure 5. Message specification modelling

```
<ServiceRequest  rdf:about="http://tapas.org/msg01">
  <sender>
    <Actor rdf:about="http://tapas.org/actorA">
      <rolePlaying rdf:resource="http://tapas.org/roleR"/>
      <nodeInstalling rdf:resource="http://comp1.tapas.org"/>
    </Actor>
  </sender>
  <dateTime>14/10/2002 GMT 14:10:00</dateTime>
  <playRequesting rdf:resource="http://tapas.org/IPM_1.0"/>
</ServiceRequest>
```

(a) ServiceRequest.

```
<ServiceComponentRequest  rdf:about="http://tapas.org/msg02">
  <sender>
     <Actor rdf:about="http://tapas.org/actorB">
       <rolePlaying  rdf:resource="http://tapas.org/printClient"/>
       <nodeInstalling rdf:resource="http://comp1.tapas.org" />
     </Actor>
  </sender>
  <dateTime>14/10/2002 GMT 15:10:00</dateTime>
  <roleRequesting rdf:resource="http://tapas.org/GraphicMaster"/>
</ServiceComponenetRequest>
```

(b) ServiceComponentRequest.

Figure 6. A service request and a service component request example.

5.3. Play configuration definition

This section presents the *play configuration definition,* comprising the following three parts: *role requirements*, *play configuration rules* and *play reconfiguration rules*. All definitions are modelled by using the *XML Declarative Description* language (XDD) (Wuwongse et al. 2001, 2003*)* and are denoted as *XDD descriptions.* A reasoning system for the Configuration Manager has been developed by means of *XET* (*XML Equivalent Transformation*) (Anutariya et al. 2002). This system can directly operate and reason about XDD descriptions.

XDD is an XML-based knowledge representation, which extends ordinary, well-formed XML elements by incorporation of variables for an enhancement of expressive power and representation of implicit information into so called *XML expressions*. Ordinary XML elements, i.e. XML expressions without variable, are called *ground XML expressions*. Every component of an XML expression can contain *XML variables*. A variable is prefixed with '\$T:' where T denotes its type. Table 1 lists the types of supported XML variables in XDD. *An XDD description* is a set of XML clauses of the form:

$$H \leftarrow B_1, \ldots B_m \{C_1, \ldots C_n\}$$

where m, $n \geq 0$, H and B_i are XML expressions. And each of the C_i is a predefined XML constraint, useful for defining a restriction on XML expressions.

The XML expression *H* is called the *head* of the clause. The set of B_i is the body of the clause. When the body is empty, such a clause is referred to an XML unit clause, and the symbol '$\leftarrow$' will be omitted. Given an XDD description *D*, its meaning is the set of all XML elements, which are directly described by and are derivable from the unit and non-unit clauses in *D*, respectively.

Table 1 Types of XML variables supported by XDD.

Type	Instantiation and examples
N	XML element or attribute names Ex: <***$N:var1***>…</***$N:var1***> can be instantiated to ***<actor>...</actor>*** or ***<node>...</node>***
S	XML string Ex: <prop name="***$S:var1***"/> can be instantiated into <prop name="***prop1***"/> or <prop name="***prop2***"/>
P	Sequence of zero or more attribute-value pairs Ex: <element ***$P:var1***/> can be instantiated into <element/> or <element ***name="1"***/>
E	Sequence of zero or more XML expressions Ex: <element>***$E:var1***</element> can be instantiated into <element/> or <element><value>***1***</value></element>
I	Part of XML expressions Ex: ***<$I:var1>***<attr/>***</$I:var1>*** can be instantiated into ***<element><prop>***<attr/>***</prop></element>***

5.3.1. Role requirement

Capability and status requirement specification of a certain role in a play is expressed as XDD descriptions. Its head specifies the role to be played, and its body describes the demanded capabilities and status of a node for fulfilling such a role. Recall that the head of an XML clause intuitively models the consequence part, while the body describes the antecedence or the conditional part. Thus, each XML clause can be easily interpreted as: deriving the information represented by its head if all the conditions specified in its body hold. Given a clause representing a role requirement specification, one can derive a list of available nodes in the network, which are capable of performing such a role. By means of CIM hierarchical schema, matching of the required capabilities and status with the offered capability and status will not only be based on exact match, but will also include a notion of reasoning through this generalisation-specialisation hierarchy. For example, if a certain role demands a computer system with a modem, knowing that PC is a subclass of computer system, and unimodem, ISDN, ADSL and cable modems are subclasses of modem, then one can derive that any PC having one of these variety types of modems has sufficient capabilities to fulfil such a requirement. Ranking of available nodes according to how closely their capabilities match with the requirements is also expressed as XML clauses. Moreover, in case there are multiple nodes satisfying the defined requirements, specification of selection preferences is also permissible by appropriate formulation of conditions in the clause's body.

5.3.2. Play configuration rules

Play configuration rules are represented as XDD descriptions. Their heads identify components of the play, while their bodies describe the configuration, composition and dependency conditions.

5.3.3. Play reconfiguration rules

Instead of providing merely a general reconfiguration mechanism, which is applicable to any trouble encountered in an application, the developed framework additionally facilitates means for definition of play specific *play reconfiguration rules*. Such rules let services encode their individual, customised reconfiguration policies, and hence allowing them to handle the same trouble in different but appropriate manners. Each time when CM receives a trouble report, it will find if there is a reconfiguration rule specifically defined for handling the given trouble or not. In case that such a rule exists, CM will generate an appropriate system reconfiguration plan according to that rule. Otherwise, the default reconfiguration, i.e., to relocate actors that are involving in the problem, will be taken place. Figure 7 defines the possible types of reconfigurations.

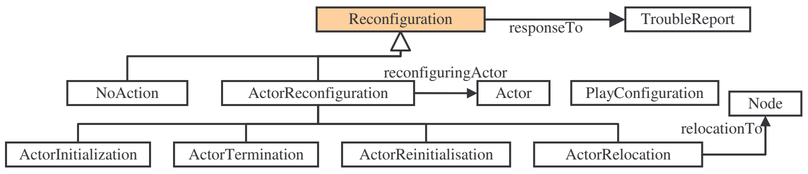


Figure 7. Reconfiguration types

The reconfiguration types are: No Action, Play Reconfiguration and Actor Reconfiguration

No Action: System developers may decide to disregard and perform no action for certain types of troubles. For instance, one may define that all actor-error reports, which involve some particular low-priority roles and are submitted during 1 AM-6 AM, will be ignored.

Play Reconfiguration: The whole running service system, defined by the specified play and consisting of multiple cooperating actors, will be reconfigured. The best node for executing each actor will be re-determined and the actor will be relocated to that new location.

Actor Reconfiguration: This requires reconfiguration of some particular service components constituted by corresponding actors in a system, and can be further classified into Actor Initialisation, Actor Termination, Actor Re-initialisation, Actor Relocation.

– *Actor Initialisation*: The action is decomposed into (i) the instantiation of a new actor at a specified node, (ii) the installation of the manuscript defining the actor behaviour which corresponds to the role to be played, (iii) the execution of the actor's operation according to the installed manuscript.

– *Actor Termination*: The specified actor will be terminated and the resources allocated to and consumed by that actor will be freed.

– *Actor Re-initialisation*: The specified actor will be terminated and re-initialised at the same node.

– *Actor Relocation*: It involves moving of an actor currently executing at one node to another. In general, this reconfiguration is carried out when an actor has insufficient capabilities to execute its functions; thus, to proceed with its operation, the actor must be relocated to a node with sufficient capabilities. The references (Shiaa et al. 2002, 2004) have already discussed how *actor mobility* is realised in TAPAS.

Reconfiguration rules are formalised by using XDD descriptions. Its head describes the reconfiguration action to be implemented, and its body represents the types, conditions and details of troubles upon which the described reconfiguration will be performed. For a given trouble report, there may exist more than one reconfiguration rule applicable to handle it. In such a case, rule prioritisation information is needed. Figure 8 summarises the data models applied in the TAPAS dynamic configuration framework.

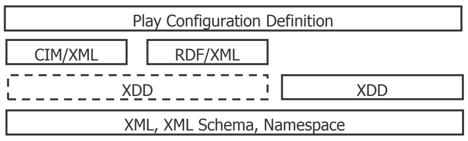


Figure 8. The Representation Layer of the Data Model

6. Demonstration: Intelligent Printing Management System

It is seen from the presented dynamic configuration architecture that CM is the primary entity which dynamically computes appropriate service (re)configuration plans by reasoning about the current system's capabilities and status, the defined role requirements, play configuration constraints and reconfiguration rules as well as the given requests and trouble reports. A prototype reasoning system for CM has been developed by means of *XET* (See Section 5.3). Here, employment of the developed architecture and the reasoning engine to, respectively, model and implement an *Intelligent Printing Management* (*IPM*) system is demonstrated along with a simple application scenario assuming the four different roles:

– *DocMaster*: a print server role for printing black & white documents.

– *GraphicMaster*: a print server role for handling colour and graphic documents.

– *IPMManager*: a role responsible for controlling and distributing print jobs to appropriate printer roles, depending on the job attributes, the current queues of each printer and the job owner privilege information. It queries and finds an appropriate print server role for executing a given job. When there exists more than one print server role capable of handling the job, a preferred one will be selected.

– *PrintClient*: an application program to which users use for sending print jobs to IPM Manager.

Note that one printer can constitute more than one print server role, and a print server role can be realised by one or more physical printers. For instance, a high-speed, laser, colour printer may be configured to play both DocMaster and

GraphicMaster roles, while the DocMaster role can be additionally realised by another black-and-white, laser printer. Moreover, in a real application scenario, there could be more varieties and more complicated types of print server roles for which different groups of users have different access controls.

When the system starts up, IPMManager and print server objects will be installed and configured. These objects receive their actual behaviour in manuscripts. Clients can be plugged in later on at any possible node running TAPAS platform. What is important from the point of view of dynamic configuration is the reasoning about these play and role requirements when installing specific roles at specific nodes.

The demonstration CM system consists of three printers (printerX, printerY and printerZ) and two computers (comp1 and comp2). printerX, printerY supports duplex black-and-white printing while printerz does not. printerX print color documents while printerY and printerZ can only print black-and-while documents. Unlike the other two printers, printerZ does not have duplex-printing capability. While comp1 is installed with Windows NT, the operating system of comp2 is Windows 98. Note that due to space limitation and for the sake of simplicity, the paper touches only the play definition for DocMaster and IPMManager. For a more complete demonstration, the reader is referred to the CM system available at http://tapas.item.ntnu.no/ipm.

6.1. Role requirement

Figure 9 gives an XDD clause C_1, formalising capability and status requirements of DocMaster role. Both graphical and textual presentation of the clause is shown. However, for ease of understanding, only graphical presentations will be used in the sequel. Recall that a variable in an XML clause is preceded with '\$', followed by its type and name. For example, **\$S**:nodeX denotes a *String*-variable named nodeX and is instantiable into only a string, while **\$E**:printerProperties is an *Expression*-variable instantiable into a list of XML expressions representing a sequence of objects or attributes. The given clause C_1 can be read as follows:

(A) An actor playing DocMaster role can be installed into \$S:nodeX, which is an instance of CIM_Printer,

if

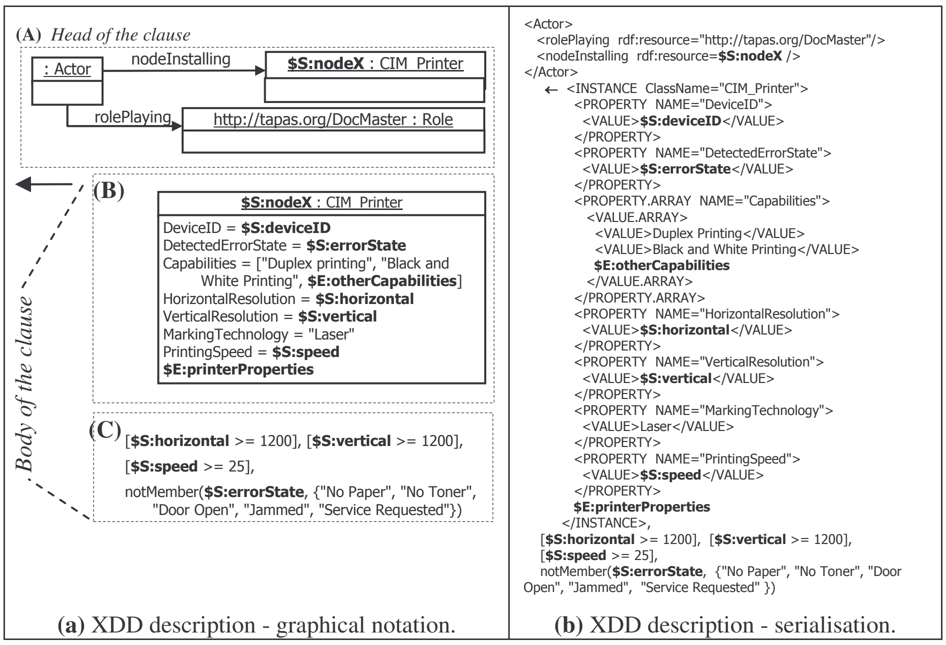
(B) \$S:nodeX is currently available and offers duplex printing and black-and-white printing capabilities, and laser marking technology,

(C) the following additional conditions on \$S:nodeX's capabilities and status are satisfied:

– [\$S:horizontal >= 1200] and [\$S:vertical >= 1200]: the horizontal and vertical resolutions of the print function are at least 1200 pixels per inch,

– [\$S:speed >= 25]: the printing speed is greater than 25 pages per minute,

– notMember(\$S:errorState, {"No Paper", "No Toner", "Door Open", "Jammed", "Service Requested"}): its current detected error state is not one of the given list.


(A) Head of the clause
: Actor
nodeInstalling
$S:nodeX : CIM_Printer
rolePlaying
http://tapas.org/DocMaster : Role
(B)
$S:nodeX : CIM_Printer
DeviceID = $S:deviceID
DetectedErrorState = $S:errorState
Capabilities = ["Duplex printing", "Black and White Printing", $E:otherCapabilities]
HorizontalResolution = $S:horizontal
VerticalResolution = $S:vertical
MarkingTechnology = "Laser"
PrintingSpeed = $S:speed
$E:printerProperties
(C)
[$S:horizontal >= 1200], [$S:vertical >= 1200],
[$S:speed >= 25],
notMember($S:errorState, {"No Paper", "No Toner", "Door Open", "Jammed", "Service Requested"})
Body of the clause

(a) XDD description - graphical notation.

```
<Actor>
  <rolePlaying  rdf:resource="http://tapas.org/DocMaster"/>
  <nodeInstalling  rdf:resource=$S:nodeX />
</Actor>
   ← <INSTANCE  ClassName="CIM_Printer">
        <PROPERTY  NAME="DeviceID">
          <VALUE>$S:deviceID</VALUE>
        </PROPERTY>
        <PROPERTY  NAME="DetectedErrorState">
          <VALUE>$S:errorState</VALUE>
        </PROPERTY>
        <PROPERTY.ARRAY  NAME="Capabilities">
           <VALUE.ARRAY>
             <VALUE>Duplex Printing</VALUE>
             <VALUE>Black and White Printing</VALUE>
             $E:otherCapabilities
           </VALUE.ARRAY>
        </PROPERTY.ARRAY>
        <PROPERTY  NAME="HorizontalResolution">
          <VALUE>$S:horizontal</VALUE>
        </PROPERTY>
        <PROPERTY  NAME="VerticalResolution">
          <VALUE>$S:vertical</VALUE>
        </PROPERTY>
        <PROPERTY  NAME="MarkingTechnology">
          <VALUE>Laser</VALUE>
        </PROPERTY>
        <PROPERTY  NAME="PrintingSpeed">
          <VALUE>$S:speed</VALUE>
        </PROPERTY>
        $E:printerProperties
     </INSTANCE>,
   [$S:horizontal >= 1200],  [$S:vertical >= 1200],
   [$S:speed >= 25],
   notMember($S:errorState,  {"No Paper", "No Toner", "Door
Open", "Jammed",  "Service Requested" })
```

(b) XDD description - serialisation.

Figure 9. Clause C1. Capability and status requirements for the Role DocMaster.

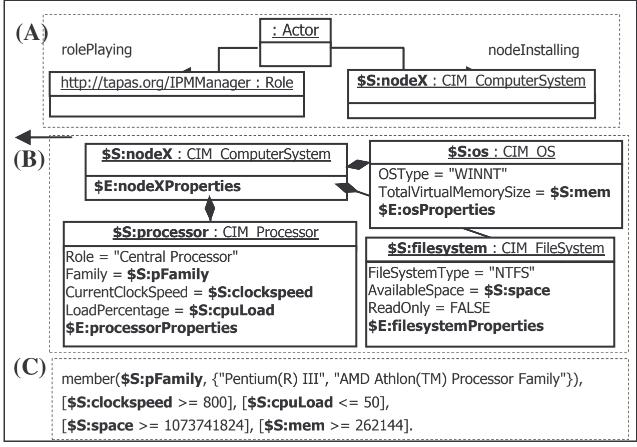
The clauses C_2 of Figure 10 gives a simple example of modelling the capability and status requirement of the role IPMManager. Since comp2 is installed with Windows 98, it cannot be selected as an IPMManager.

6.2. Play configuration and reconfiguration rule

Figure 11 and Figure 12 present the play configuration and reconfiguration rules of the demonstrated IPM system, respectively.

6.3. Computing play configuration and reconfiguration plans

The play configuration definition for the IPM system is modeled my XDD descriptions, comprising the requirement specification of each role in a play as well as the play configuration constraints and the reconfiguration rule. In addition, CSRep comprise CIM instances maintaining the capabilities/status of current, available nodes in a system is needed as input to CM. Assume that the CM receives the ServiceRequest of Figure 6.a for installing and configuring the IPM system. A configuration plan computed by the CM by means of the prototype reasoning system is illustrated by Figure 13. It specifies that: (i) an actor playing the role IPMManager is to be installed at comp1, (ii) the role DocMaster at printerX and printerY, and (iii) the role GraphicMaster at printerX.

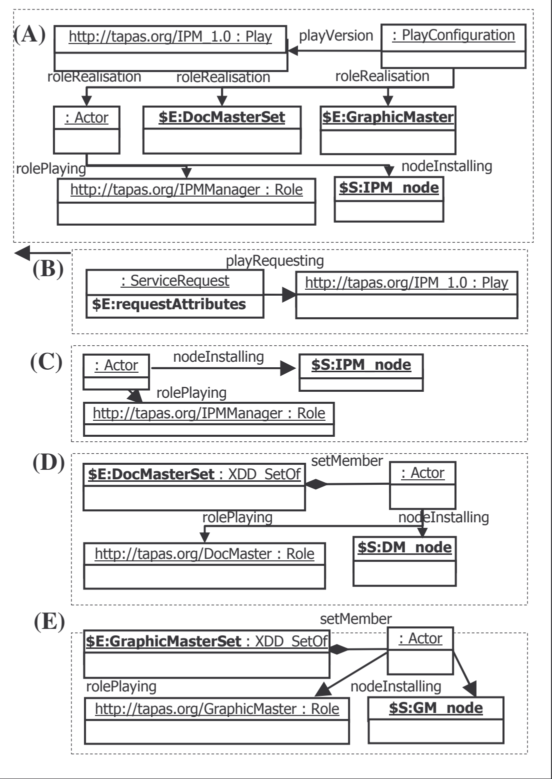

(A)
: Actor
rolePlaying
nodeInstalling
http://tapas.org/IPMManager : Role
$S:nodeX : CIM_ComputerSystem
(B)
$S:nodeX : CIM_ComputerSystem
$E:nodeXProperties
$S:os : CIM_OS
OSType = "WINNT"
TotalVirtualMemorySize = $S:mem
$E:osProperties
$S:processor : CIM_Processor
Role = "Central Processor"
Family = $S:pFamily
CurrentClockSpeed = $S:clockspeed
LoadPercentage = $S:cpuLoad
$E:processorProperties
$S:filesystem : CIM_FileSystem
FileSystemType = "NTFS"
AvailableSpace = $S:space
ReadOnly = FALSE
$E:filesystemProperties
(C)
member($S:pFamily, {"Pentium(R) III", "AMD Athlon(TM) Processor Family"}),
[$S:clockspeed >= 800], [$S:cpuLoad <= 50],
[$S:space >= 1073741824], [$S:mem >= 262144].

The clause specifies that:

(A) Any instance of the class CIM_ComputerSystem, represented by $S:nodeX, can install an actor for playing the role IPMManager, if the requirements (B) and (C) defined by the clause's body are met. That is:

- The central processor of that $S:nodeX is either Pentium III or AMD Athlon families with 800-MHz minimum clock speed and the load percentage less than 50.
- The installed operating system is WINNT with the minimum virtual memory of 262144 KB (256 MB).
- The computer's hosted file system must be NTFS with available space at least 1073741824 bytes (1 GB).

Figure 10. Clause C2. Capability and status requirements of the Role IPMManager



An XML clause defining a configuration rule of the play http://tapas.org/IPM_1.0.

The head (A) expresses that a valid configuration comprises the realisation of certain roles in the play, specified by three roleRealisation associations. The first association indicates that there must exist exactly one actor constituting the role IPMManager at a node $S:IPM_node. The other two associations, relating to the objects $E:DocMasterSet and $E:GraphicMaster, specify that the configuration also contains installation of actors realising some particular roles. The conditions on the number of actors to install and the roles to play are defined by the clause's body.

The clause's body, comprising (B)–(E), specifies conditions for derivation of the defined configuration as well as its composition structure.

(B) indicates that the configuration will be computed upon the receipt of a ServiceRequest for installing such a version of the play.

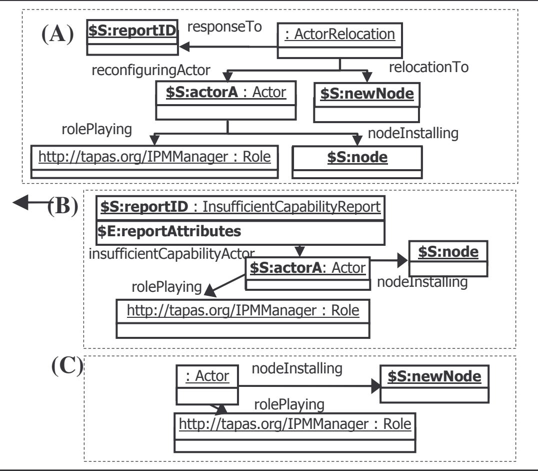
(C) ensures that the node represented by $S:IPM_node has sufficient capabilities and status to install an actor for executing the role IPMManager. Obviously, this expression will be matched with the head of the clause C_2 of Figure 10, which models the IPMManager's requirement.

(D) specifies that $E:DocMasterSet represents a set of actors to be installed at nodes that are capable of playing the role DocMaster. That is, these nodes' capabilities and status must meet the requirement of the role represented by the clause C_1 of Figure 9. In this example, there can be more than one actor realising the role DocMaster.

Similarly, (E) specifies that $E:GraphicMasterSet represents a set of nodes capable of playing the role GraphicMaster.

Note that a restriction on the number of roles that an instance can play is not defined. A particular printer may realise both DocMaster and GraphicMaster roles at the same time.

Figure 11. Clause C3. A play configuration rule of the IPM service system.



The clause models a specific reconfiguration rule for handling InsufficientCapabilityReport of an actor playing the role IPMManager.It defines that:

(A) an ActorRelocation plan, specifying that the actor $S:actorA is to be relocated to $S:newNode, will be derive,

if

(B) there arises an InsufficientCapabilityReport, identified by $S:reportID and describing that the actor $S:actorA, currently playing the role IPMManager at the node $S:node, has insufficient capabilities to execute its functionality, and

(C) there exists a node in the system which is currently available and capable of playing the role IPMManager, and denote such a node by $S:newNode.

Figure 12. Clause C4: A dynamic reconfiguration rule of the IPM service system.

(a) The generated XML document representing the computed plan.

```
<PlayConfiguration>
  <playVersion rdf:resource="http://tapas.org/IPM_1.0"/>
    <roleRealisation>
      <Actor>
        <rolePlaying rdf:resource="http://tapas.org/IPMManager"/>
        <nodeInstalling rdf:resource="http://comp1.tapas.org"/>
      </Actor>
    </roleRealisation>
    <roleRealisation>
      <Actor>
        <rolePlaying rdf:resource="http://tapas.org/DocMaster"/>
        <nodeInstalling rdf:resource="http://PrinterY.tapas.org"/>
        <nodeInstalling rdf:resource="http://PrinterX.tapas.org"/>
      </Actor>
    </roleRealisation>
    <roleRealisation>
      <Actor>
        <rolePlaying
            rdf:resource="http://tapas.org/GraphicMaster"/>
        <nodeInstalling
            rdf:resource="http://PrinterX.tapas.org"/>
      </Actor>
    </roleRealisation>
</PlayConfiguration>
```

(b) Corresponding RDF graph.

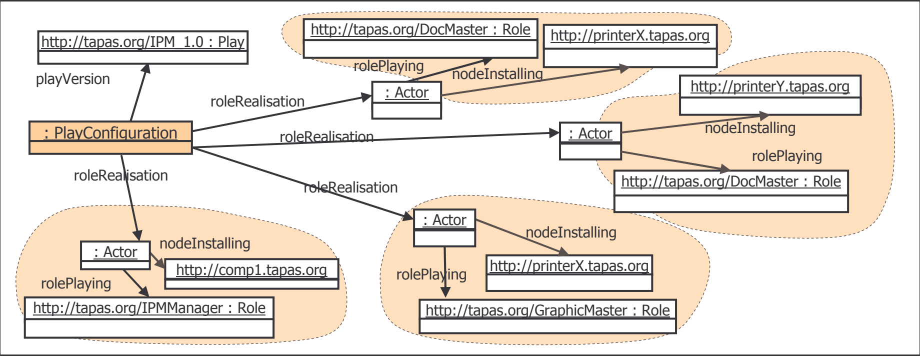

http://tapas.org/IPM_1.0 : Play
playVersion
: PlayConfiguration
roleRealisation
roleRealisation
roleRealisation
roleRealisation
http://tapas.org/DocMaster : Role
http://printerX.tapas.org
rolePlaying
nodeInstalling
: Actor
http://printerY.tapas.org
: Actor
nodeInstalling
rolePlaying
http://tapas.org/DocMaster : Role
: Actor
nodeInstalling
http://comp1.tapas.org
rolePlaying
http://tapas.org/IPMManager : Role
: Actor
nodeInstalling
http://printerX.tapas.org
rolePlaying
http://tapas.org/GraphicMaster : Role

Figure 13. Calculated configuration plan for the IPM service system.

7. Conclusions

A uniform representational and reasoning framework for dynamic configuration of service systems in TAPAS architecture has been developed, and its employment to model an Intelligent Printing Management system has been demonstrated. The framework enables services to be composed on the fly and the location for executing service components to be determined dynamically based on the offered capabilities as well as the current situation in the network. Moreover, during the service execution, it also permits adaptation of the service composition structure if certain significant events, such as a service component failure or QoS degradation, occur. In the framework, the Configuration Manager is the primary entity which reasons about the current system's capabilities & status, services' requirements and reconfiguration policies in order to dynamically generate appropriate service (re)configuration, hence enabling the system to cope with variations in the environment, achieve mandated performance levels and meet user satisfaction. To verify the framework's feasibility and potential in real applications, it has been implemented using the XET reasoning engine.

References

Aagesen, F. A., C. Anutariya, et al. (2002). Support Specification and Selection in TAPAS. IFIP WG6.7 Workshop and Eunice Summer School on Adaptable Networks and Teleservices, Trondheim, Norway, Tapir.

Aagesen, F. A., B. E. Helvik, et al. (2003). On Adaptable Networking. Int'l Conf. on Information and Communication Technologies (ICT 2003), Assumption University, Thailand.

Aagesen, F. A., B. E. Helvik, et al. (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.

Aagesen, F. A., B. E. Helvik, et al. (1999). Towards a Plug and Play Architecture for Telecommunications. 5th IFIP Conf. Intelligence in Networks (SmartNet 99), Bangkok, Thailand, Kluwer Academic Publisher.

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

Bakour, H. and N. Boukhatem (2004). ASMA: An Active Architecture for Dynamic Service Deployment. IFIP Int'l Conf. Intelligence in Communication Systems (INTELLCOMM 2004), Bangkok, Thailand.

Berners-Lee, T., M. Fischetti, et al. (1999). Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor, Harper, CA.

Bieszczad, A., B. Pagurek, et al. (1998). "Mobile Agents for Network Management." IEEE Communications Surveys **1**(1).

Brickley, D. and R. V. Guha (2004). RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004. B. McBride.

Cohen, R. and D. Raz (2004). An Open and Modular Approach for a Context Distribution System. Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea.

D'Antonio, S., M. D'Arienzo, et al. (2004). An Architecture for Automatic Configuration of Integrated Networks. Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea.

Hendler, J. and D. McGuinness (2000). "The DARPA Agent Markup Language." IEEE Intelligent Systems **15**(2): 72-73.

Inoue, Y., M. Lapierre, et al. (1999). The TINA Book: A Co-operative Solution for a Competitive World, Prentice Hall.

Keller, A., J. L. Hellerstein, et al. (2004). The CHAMPS System: Change Management in Planning and Scheduling. Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea.

Lassila, O. and R. R. Swick (1999). Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999.

McGuinness, D. L. and F. Harmelen (2004). OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004.

Raza, S. K. and A. Bieszczad (2004). Network Configuration with Plug and Play Components. Proc. 6th IFIP/IEEE Network Operations and Management Symposium (NOMS 2004), Seoul, Korea.

Sahai, A., S. Singhal, et al. (2004). Automated Policy-Based Resource Construction in Utility Computing Environments. IEEE/IFIP Network Operations and Management Symposium NOMS'2004, Seoul, South Korea.

Shiaa, M. M. (2004). Mobility Support Framework in Adaptable Service Architecture. IEEE/IFIP Net-Con' 2003, Muscat, Oman.

Shiaa, M. M. and F. A. Aagesen (2002). Mobility Management in Plug and Play Network Architecture. Proc. IFIP 7th Int'l Conf. Intelligence in Networks (SmartNet 2002), Saariselka, Finland, Kluwer Academic Publishers.

Shiaa, M. M., S. Jiang, et al. (2004). An XML-based Framework for Dynamic Service Management. The 2004 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 04), Bangkok, Thailand.

Solarski, M., L. Strick, et al. (2004). Flexible Middleware Support for Future Mobile Services and Their Context-Aware Adaptation. IFIP Int'l Conf. Intelligence in Communication Systems (INTELLCOMM 2004), Bangkok, Thailand.

Tennenhouse, D. L., J. M. Smith, et al. (1997). "A Survey of Active Network Research." IEEE Communications **35**(1).

Westerinen, A. and J. Strassner (2000). "Common Information Model (CIM) Core Model Distributed Management Task Force White Paper Version 2.4."

Wuwongse, V., K. Akama, et al. (2003). "A Data Model for XML Databases." Intelligent Information Systems **20**(1): 63-80.

Wuwongse, V., C. Anutariya, et al. (2001). "XML Declarative Description (XDD): A Language for the Semantic Web." IEEE Intelligent Systems **16**(3): 54-65.