Reconfiguration Management in Self-Organizing Systems beyond 3G

Athanasia Alonistioti, Christos Anagnostopoulos, Gerasimos Stamatelatos
Communication Networks Laboratory, Department of Informatics & Telecommunications, University of Athens, Greece

Email: {nancy, bleu, makiss}@di.uoa.gr

Abstract - ***The main goals for the introduction of next generation mobile systems are the integration of software based communication concepts, provision of broadband access, seamless global roaming and Internet/Data/Voice everywhere, utilizing each time the most "appropriate" technology. Further, such a dynamic environment will enable the delivery of situation-aware, personalized multimedia services over heterogeneous, ubiquitous infrastructures. The development, delivery and management of mobile services are the subject of many research activities in both the academia and industry. Reconfigurability and adaptability are key aspects of the mobile systems beyond 3G. In addition, reconfigurable mobile systems and networks introduce additional requirements and complexity. This paper presents an object oriented reconfiguration management metamodel and a number of patterns that aim to model the abovementioned aspects*.

Keywords: Reconfigurability, Reconfiguration Management, Self-Organizing Systems, Metamodel.

I. Introduction & Related Work

The issue of reconfigurability has been tackled in the past mainly in the two edges of the OSI layer model, namely the physical and the application. The physical layer related research has been carried out so that devices can detect and use the available networks. However, the research was limited to the use of different physical layers to carry the information and no provision was made for the interoperability with the application's requirements. Furthermore, several attempts have been made for the introduction of adaptive protocols and respective design ([7]). Building on the knowledge, from early software radio projects in the military domain, SDR Forum has pioneered in exploring reconfigurability concepts in the United States. However, being the vanguard of reconfigurability developments and the first to define a software radio architecture [3] [4], seems to have come at the expense of a rather focused view on reconfigurability that addresses primarily the radio domain (RF processing, down-conversion, RF processing, A/D conversion, etc) [4]. On the application layer, research has been carried out on the adaptation of the application or service according to the predefined profiles of the user and the service in the MOBIVAS [5] platform [6]. The user can discover different instances of the service according to the profile and the terminal capabilities of his device. In the IST-TRUST and SCOUT projects mobility and radio resource management issues have been addressed [7]. Recently, OMG has introduced the SBC group (Software Based Communications), which addresses issues relevant to the integration of software technologies to serve the solutions for flexible communication systems [5]. Issues related to Software Radio have been addressed in the joint SWRADIO FTF and SBC DTF meeting where OMG adopted a draft PIM and PSM SWRadio Components specification [8]. Joe Mitola has pioneered the concept of Software Radio and the integration of object oriented technologies to support flexible communication and cognitive systems [9], [10], [11]. The tackling of the problem, since it was mainly in the two edge layers, physical and application is neither efficient nor sufficient; therefore it cannot provide a robust solution for achieving reconfiguration end-to-end. Based on the above discussion, it is apparent that in the design of fully reconfigurable networks and systems, the introduction of advanced reconfiguration management functionality is necessary. A holistic solution for addressing reconfiguration management across all layers is introduced as a Reconfiguration Management Plane (RMP). RMP enhances reconfigurability control in order to address end-to-end reconfiguration management aspects [2]. In this paper we introduce a meta-model for reconfiguration related functionality specification and a number of patterns that evaluate its significance in a mobile computing environment.

The rest of this document is structured as follows: Section II depicts our conceptual pattern elements using the Unified Modelling Language (UML) and represents it through a class diagram. Section III defines such model via the reconfiguration stereotypes, to evaluate the different parameters of the reconfiguration algorithm (e.g., class-marks, triggers, and context). Section IV refers to value added service creation pattern that is related to service provision concepts (e.g., user profiles and terminal capabilities). Section V introduces the personalized service provision pattern and describes semantics with a certain usage of user preferences. Section VI analyses the reconfiguration interpretation of the defined patterns, indicating the role of reconfiguration. Finally, conclusions and directions for further work in the area are provided in Section VII.

II. Definition Of Reconfiguration

We could envisage the notion of Reconfiguration as an abstract process, which is based on how to efficiently adapt, apply and upgrade the functionality that an entity supports, to any expected or potential change/alteration of its state, situation and activity. In the basic reconfiguration scheme, as presented in Figure 1 an Actor reconfigures a Reconfigurable Entity composed by a certain number of discrete Manageable Elements. Furthermore, the Actor maintains a "manipulates" relation with each of these Manageable Elements exploiting their functionality. Such an abstract definition is further described by introducing several patterns and model elements that interpret the notion of reconfiguration.

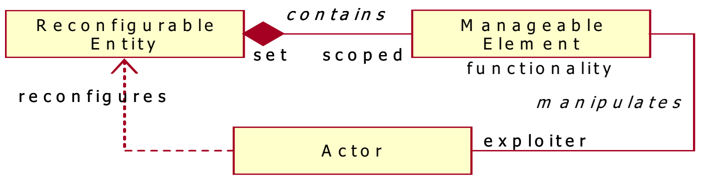


Figure 1. Essential Reconfiguration Scheme

III. Pattern Elements

We refer to model elements that act in certain spaces of a Reconfigurable environment. The spaces are predefined scenarios into which model elements unfold their capabilities and communicate with each other appropriately. A reconfiguration action derives the functionality of such model elements and is aggregated to specific algorithm in order a service perception or reconfiguration event to be carried out. Such elements are assigned different roles in different spaces. An element may be engaged to an activity for a certain space, but it also may have been assigned another role in the same or in a different space. In the UML syntax, the concept of inheriting different roles in different spaces is modeled as *taggedValues.* Every element attributes to predefined model pattern that maintains a set of semantically ordered taggedValues, which address the role and any additional information that an element provides in a space.

The aforementioned patterns extend the *PatternClass*, *PatternBehavior* and *PatternOperation* stereotypes. PatternClass is referred to as any function or algorithm, which plays different role into different spaces (i.e., the multiple taggedValue pattern, with space-name being set as "name" and role-name being set as "role"). PatternBehavior stands for an activity that characterizes an element, such as an open interface or the methods of an object. Further, PatternOperation stands for an operation, which is fired/activated whenever certain rules are applied (i.e., a decision maker or a trigger object may implement a set of such operations).

In addition, the relations among such elements are fully defined in a steady state once they are envisaged as stereotyped Dependency. The sole dependency among them is collectively called *Reconfigurability,* indicating that an element is reconfigured by another or by itself (i.e. the degree of decision making and self reconfiguration capabilities characterizes the concepts of cognitive and self-organizing elements). We further refer to *Constraints* among elements. Such Constraints restrict or control the functionality of the elements, which can be envisaged as a special kind of policies that a decision-making mechanism is abided by. The elements are coefficients of stereotypes' construction and support the pattern-based design.

IV. Reconfiguration Stereotypes

We firstly introduce a set of stereotypes that declare the behavioral attributes of the model elements. The stereotype *Provision* is defined as a generic concept of service perception to anyone that asks for it. A provision space may be secure and may take into consideration the *mobility* pattern of the requester. The mobility concept is just a trigger that labors the provision process (i.e., service downloading for mobile users). Such attributes are defined as taggedValues indicating the extension concept of the PaternClass. The main stereotype of all the spaces is referred to as Reconfiguration. Such stereotype inherits both the PatternClass and the Provision stereotyped concepts and stands for an algorithmic and functional point of view. The reconfiguration concept is more extended than the provision one and is marked as transparent or not (i.e., transparent taggedValue). Dependencies among atomics that are stereotyped as Reconfiguration are also marked as Reconfigurability. This dependency illustrates the strong relationship among reconfiguration atomics that are due to implement any reconfiguration process or algorithm (i.e., patch downloading, protocol replacement, dynamic service provision).

Reconfigurability, as semantic dependency, may be implemented as priority or time ordered messages from context detection atomics (i.e., network monitoring systems, context managers, sensor network information accumulators) to decision-making atomics. Certain triggers that maintain rule-based like policies indicating the specific time-stamp for triggering any decision-making system fire this messaging scheme. Such atomics are stereotyped as *Triggers* and are assumed to extend the operational nature of any reconfiguration algorithm. The Trigger performs its task with a predefined intention (i.e., the taggedValue description), which it may be an urgent or an optional event. For instance, a terminal handover process may trigger the running service to be adaptable to such change by requesting from the base station the appropriate protocol downloadable component.

The rules that stand for any kind of triggering may be explicitly described and interpreted by every triggering atomic. The stereotype *policy* is marked describing the case in which there exists an interoperable and certifiable model for a policy-like model among heterogeneous reconfiguration systems. The policies form the main construction of knowledge that includes every rule for performing such algorithms (e.g., from the implantation point of view such rules cover the business logic of a distributed reconfiguration framework).

The piece of system information that refers to the temporal or static description of the objects is maintained into *profiles*. Any atomic that holds the permanent or contextual information about itself is self-described by the use of well-predefined profile models. The profile atomic may be referred to user, equipment, service, security, charging, or network elements each of which forms a meaningful object for time and/or activity oriented status (i.e., context) monitoring. The profile tags a specializer (i.e., the monitored element), a local attribute that indicates whether such information is distributed among systems or is located physically in a single system, (e.g., profile repositories) and static flag that signs the dynamic nature of the stored information. A static profile stands for the steady attributes of the profile and a dynamic profile is formed by the dynamic alteration of such attributes. Section IV details the profile contextual information.

Once the profile atomic is a specializer of any element in mobile computing environments, the notion of classification of the reconfiguration feasibility that an element shares for is of high importance. The definition of a *classmark* as a classification attribute is mandatory. Such aggregated information feeds the decision making atomic to perform intelligent actions. The classmark stereotype is a pattern class indicating how "reconfigurable" an element may be and whether certain actions may be performed, causing certain alterations to the element's functionality. The tag classmark for a certain element is constructed by information that is gathered and collated by the different sections of profiles, the various patterns describing the result of any applied reconfiguration algorithm, and the contextual or behavioral history. The reconfiguration capabilities are inherent part of this classmark, in order to enable the characterization of the degree of reconfiguration the element or e.g., equipment can undergo. The classification of an element related to the classmark sign may conclude (not absolutely) the following list:

- Static classmark: Low level of reconfiguration capability. The element (e.g., user equipment) may need to reboot its software component in order to be adaptable to any request for reconfiguration.
- Quasi-static classmark: A middle static capability feature (e.g., the running service may interrupt its connection but the user equipment is just requesting for a better level of bandwidth).
- Quasi-dynamic classmark: A middle dynamic capability feature (e.g., the running service is adaptable to any change).
- Dynamic classmark: Fully reconfigurable element (e.g., the user equipment may negotiate and take decisions about its contextual state).

We introduce the role and the spaces of a Reconfiguration environment as the service provision pattern and we extend such functionality to the fully reconfiguration pattern.

V. Value Added Service Creation Pattern

This section describes the service creation pattern from a business model point of view. A value added service development involves several roles, namely, the *ApplicationProvider*, the *ContentProvider*, the *ServiceProvider*, and the *ValueObject*. Each of the roles has been modeled as class, implementing the already described patterns. Figure 3 depicts the space of this pattern.

More specifically, the *ValueObject* class interprets and captures the result of a value creation process (e.g., the software element of the downloadable service). This pattern illustrates a step-by-step value creation involving different roles and strong sense of specialization .The value creation becomes more efficient once the involved roles specialize in the part of the value creation process where they have the greatest competence. In this case, the ValueObject is differentiated according to the specialization of each individual role being involved in every value creation process.

The ApplicationProvider class, stereotyped as provider, models the role of software developer as a step of the composite value creation process. The created value stands for the developed application, as declared by the attribute application attributes to "ValueObject". The corresponding role is further described by the "creates" realization that exists between ApplicationProvider and ValueObject classes. The secure tagged value is set "true" indicating the requirement that every single application development process must be carried out with compliance to the adopted security constraints (i.e., security policies are applied to any ValueObject concept). Additionally, the mobility tagged value is set "true" indicating that the described value creation takes into account the potential user's mobility behavior. The ContentProvider class, stereotyped as provider, models the role of content repositories and maintenance as collectively declared by the realization “maintains” existing between ContentProvider and ValueObject classes. The created value, as a step towards the composite value creation, stands for each autonomous unit of content,, which is stored within the aforementioned repositories,. The secure tagged value is also set “true” indicating that a content unit development and maintenance process must be carried out in compliance with the adopted security policies. Additionally, the mobility tagged value is set “false” indicating that the described value creation has no further relevance to the user’s mobility behavior.

Finally, the ServiceProvider Class, also stereotyped as provider, materializes the value added service provision role. ServiceProvider, by composing the corresponding values created by the ApplicationProvider and ContentProvider classes, finalizes the value creation. This composing process is indicated by the application and content attributes set to ValueObject as well as the “cooperates” bi-directional associations that exist between ServiceProvider and the rest of classes. Furthermore, such a finalization is indicated by the “obtains/provides” realization that links ServiceProvider and ValueObject.. The “true” secure tagged value is set “true” indicating that the service provision process must be accomplished in secure transaction. The ServiceProvider’s mobility value must be the same with ApplicationProvider’s one. Setting the mobility value “true”, indicates a full service provision with respect to user’s mobility features. On the other hand, ServiceProvider is not aware of the content adaptation regarding the user’s mobility. This is indicated this by setting the ContentProvider’s mobility to opposite value.

VI. Personalized Service Provision Pattern

This section describes the so–called Personalized Service Provision Pattern that attempts to model the various service provision processes (the SP0 space refers to personalized service provision). A service provision process (i.e., service perception space), is carried out in a personalized way, by considering set of certain activities that involve context management, profile management, software development, charging and privacy related policies.

In the beyond 3G telecommunications systems , the requirement for anywhere and any terminal, personalized access to services is to be of high importance. By consuming flexible service provision tasks, mobile users evaluate a single access point, through which the discovery and optimal selection of a plethora of services are performed and can be tailored to the service provision context (i.e., terminal capabilities, user location, and network characteristics) as well as personal preferences [1].

The Personalized Service Provision Pattern is based on a lightweight UML profile that models the various interactions among certain roles, involved in the provision process.

The PersonalizationServiceProvider being stereotyped by the Provision concept interprets the main role for the proposed personalized service provision action. The tagged value *secure* is set “true” declaring that security features assumingly are applied in every individual service provision action, whilst the *mobility* one is set “true” declaring that the certain mobility pattern (i.e., mobility model, path prediction algorithms) must be taken into account in such process. Such tagged values have already described in Section III. PersonalizationServiceProvider is the final adopted role in the service provision process that provides a value added service to end-users as a result to various interactions and individual processes. Such interactions and processes are described in this paper as follows.

The *UserInfo* class, stereotyped as *Profile*, models information that is related to the end user (i.e., consumer of the service or VASP), having been indicated by the *specializer* tagged value by setting it to “user” and the local one to “true”. Such information stores, personal data for user identification (e.g., authorization data) as well as description of user’s charging capabilities. The modelled information may be either constant over the time or changeable. Commonly, the *UserPreferences* class is stereotyped as profile and maintains more specific information referred to a single user as indicated by the specializer tagged value being set to “user” and the local one being set to “true”. Such information may further hold the user’s cost related preferences (e.g., charging profiles that may be related to a maximum amount of money that a user is willing to pay for a service. This information is characterized to be dynamic, as the tagged value static is set to “false”. In this case, such information may be obtained by monitoring procedures and changes rapidly. The *UserProfile* class, stereotyped as profile, manages the end users’ profiles. Figure 4 illustrates the UserProfile class as a composition of the corresponding UserInfo and UserPreferences classes, by acting as profiles aggregator (i.e., this is declared by the tagged value static set as “false”). The specializer tagged value is set to “user”, indicating how the specific type of profile is managed. Furthermore, the static tagged value set to “false” indicates that the managed information is dynamic.

The *TerminalCapabilities* class models the profile of specific user equipment as the static value local set to “true”. More specifically, such a profile is composed by the equipment’s attributes and capabilities, such as processing, storing and displaying capabilities and is essential for personalization purposes in service provision processes. TerminalCapabilities is stereotyped as Profile”. The tagged value static is set to true, declaring that the user equipment characteristics are either constant over the time or change in a long-term basis.

The *ServiceDescription* class models the profile of a specific value added service as declared by the tagged value specializer that is set to “service” and the local one that is set to “true” is stereotyped as Profile. A service profile is composed by characteristics and constraints of the service such as qualifier, vendor, QoS indicators, and charging information. Such information, which is of static nature, is indicated by the static tagged value set to “false”.

The *Privacy* class, stereotyped as Policy*,* models the specific regulation context, regarding the adapted privacy policies (i.e. traceability, non repudiation, and secure transactions). The *isDescriptive* tagged value is set to “true” indicating that the current privacy policies exist in a descriptive way (i.e., declarative policy languages).

The *Billing* class, stereotyped as Policy, models the current accounting, charging, and billing schemes that operate in any service provision process. The *PersonalizationProfileManager* class, stereotyped as *Provision*, maintains a *personalizer* role. Such role is a significant one in every service provision process as it initiates and coordinates the main action of personalization. As illustrated in Figure 4, PersonalizationProfileManager manages the static profiles acting as aggregator. The performed management is closely related to gathering the required information leading the personalization action and must be carried out in a secure way, as indicated by the tagged value *secure* set “true”. Furthermore, the tagged value *mobility* is set to “false” and is related with the management for only static profiles.

On the other hand, the *PersonalizationContextManager*, also stereotyped as Provision, maintains a *context* role. More specifically, PersonalizationContextManager manages the dynamic profiles that are involved in a service provision process, acting also as an aggregator. The management relates to capturing and fusion of the dynamic-nature information providing some necessary feedbacks. The secure tagged value is also set to “true”, indicating the security features of the performed actions. The mobility tagged value is also set to “true”, declaring that this class aggregates and manages dynamic profiles. PersonalizationContextManager maintains a role, which is complementary to PersonalizationProfileManager’s one, as indicated by the “interacts” bi-directional association that links the aforementioned classes.

Finally, such class is the decision-making model element. It obtains the required information for a personalized service provision communicating with the PersonalizationProfileManager. Similar interactions are described by the coherent cooperation of the bi-directional association rules.

VII. Reconfiguration Pattern

We introduce a reference model that relates to distinct meta-model description of a reconfiguration supportive system. Such system conforms to certain reconfiguration patterns into which well-defined spaces, as depicted in Figure 5, (i.e., RC0 space refers to informational driven reconfiguration and RC1 space refers to activity driven reconfiguration) assign roles to any element. We firstly, meet the notion of a *ContextManager* that gathers and collectively composes relevant contextual information of distributed resources. Such information relates to mobility data and to history actions when it is referred to temporal data collation, and to profile information when it is referred to atomic's information. The former informational set of resources are stereotyped as *PatternBehavior* because indicate the information generated by performed activities of any element, such as user's service downloads, network elements status and protocols states. The latter sets of resources are stereotypes as profiles and maintain the time specific value of such sets.

The ContextManager composes such information in order to feed and initiate any trigger action defined and interpreted by the *RCTrigger* component (i.e., stereotype Trigger with triggering intention as a "notifier"). Such trigger coefficient just notifies the system about any contextual alteration took place related to an element (e.g., terminal status) or to a cluster of elements (e.g., terminals connected via the same network protocol). Applying provision algorithms may initiate and render for provision actions that in turn they could lead to reconfiguration actions monitored by Trigger atomics. The RCTrigger fires the rules of the reconfiguration knowledge base described in policy logics. The fired actions are declared in the *RCAction* Trigger and inform appropriately the decision-making component incorporated into the *ReconfigurationManager* (maybe via certain communication protocol forming the reconfiguration control signaling). Such manager retains a "Reconfigurability" dependency with its monitored *LocalReconfigurationManager*s that is considered the overall supervisor of any reconfiguration action. Its monitored managers, the so-called LocalReconfigurationManagers (LRMs) are responsible for achieving and implementing the reconfiguration commands and algorithms to any *Component* that are attached. The LRM maintains partial knowledge of the attached Component (e.g., terminal, base station) and controls the downloading process incorporating the software-downloading manager *SWDownloadManager*. The latter retrieves the meta-data of the downloadable software and the software itself. Such data could be maintained to profile manager's repository distributed or aggregated to certain computation space. The Component that is supervised by an LRM may be reconfigurable or not and respectively may be classified according to its reconfigurability capability by certain *ClassificationStub* (i.e., the collectively called classmark). Gathering information from the Component's profile, the runtime Component's context, its classmark tag, and any others Components' information related with the supervised one generates any action, which the associated LRM has to apply.

VIII. Use Case for Reconfiguration

In this scenario (see Figure 6), the case of a malfunctioning user terminal is being considered. The Manufacturer undertakes the RCTrigger role and develops a new parch, with reference to the value creation process that has been described within Section V. The Reconfiguration Control Manager (RCM), that undertakes the ReconfigurationManager role, has implemented a certain interface enabling Manufacturer to provide patch management. When Manufacturer registers its new patch, the Reconfiguration Control Manager (RCM) queries ContextManager with the appropriate knowledge about the patch for a list of the user terminals that are capable of performing the corresponding upgrade. ContextManager derives the requested list by filtering the appropriate contextual information and responds to RCM that notifies the corresponding LRM(s), allocating the reconfiguration management. Each LRM triggers DownloadManager to manage/lead every single downloading process, and the patch in question is securely downloaded to the corresponding user's terminal, without any interference by the user side. Finally, each user terminal installs the downloaded patch.

Conclusions and Future Work

The evolution of reconfigurability notion has been heralded as main concept for 4G mobile communications. In order to reach its full potential, a consistent framework that deals with reconfigurability challenges and control has to be introduced. In this paper we have introduced a reference model which materializes these challenges, and in the future our work will focus on extending the described metamodel and model using more specific models. Such model develops a generic framework to cope with the complexity of reconfigurability management. This work will provide the basis for the evolution of End-to-End Reconfigurability notions. The proposed model for reconfiguration management addresses the effective policy based reconfiguration triggering towards the network nodes and the combination of adaptation triggering towards the end-user services in order to achieve the optimal service provision and perception to the end user in a transparent way. Our research focuses on innovative profile definitions and interpretations, such as network profile, which they are assumed to enrich the reconfiguration context deployed to a holistic decision making mechanism for further efficient reconfigurable applications.

Acknowledgement

This work is partially funded by the Commission of the European Communities, under the 6th Framework Program for Research and Technological Development, within the project End-to-End Reconfigurability (E2R). The authors would like to acknowledge the contributions of their colleagues from E2R consortium.

References

[1] Athanasia Alonistioti, Fotis Foukalas, and Nikos Houssos, "Reconfigurability management issues for the support of flexible service provision and reconfigurable protocols"*, proc. SDR Forum 2003 technical conference,* November 2003, Orlando, Florida

[2] N. Alonistioti, A. Glentis, F. Foukalas, and A. Kaloxylos,"RMP: Reconfiguration Management Plane for the support of Policy Based Network Reconfiguration"*, proc. IEEE PIMRC 2004*, September 2004, Barcelona, Spain.

[3] The Software Defined Radio Forum, http://www.sdrforum.org/

[4] The Object Management Group, http://www.omg.org

[5] IST-MOBIVAS project, http://mobivas.cnl.di.uoa.gr

[6] N. Houssos, V. Gazis, A. Alonistioti, " Application transparent Adaptation in wireless systems beyond 3G", *2nd International Conference on Mobile Business (M-Business 2003)*, Vienna, Austria,2003

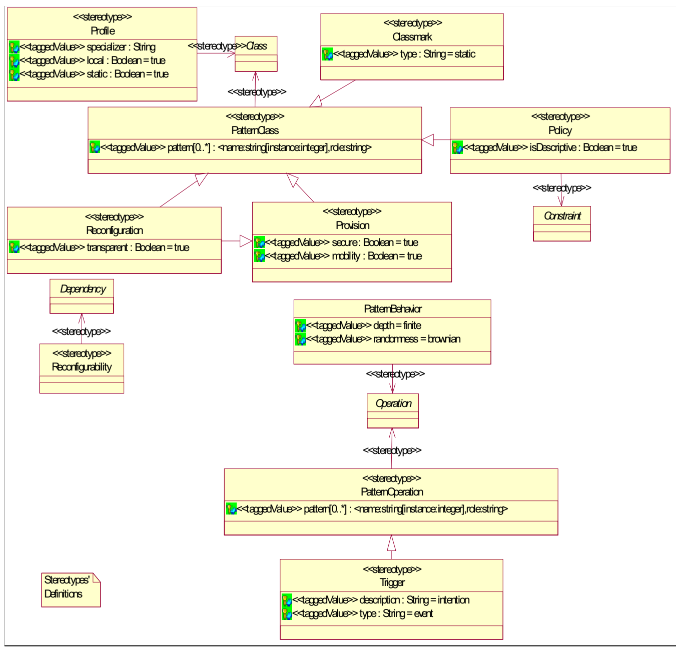
[7] M. Dillinger, K. Madani, N. Alonistioti, "Software defined radio, Architectures, Systems and Functions", John Wiley & Sons, Ltd ISBN: 0-470-85164-3

[8] http://www.omg.org/docs/dtc/04-05-04.pdf

[9] "Software Radio Architecture: Object-Oriented Approaches to Wireless Systems Engineering", Joseph Mitola, ISBN: 0-471-38492-5, October 2000, Wiley.

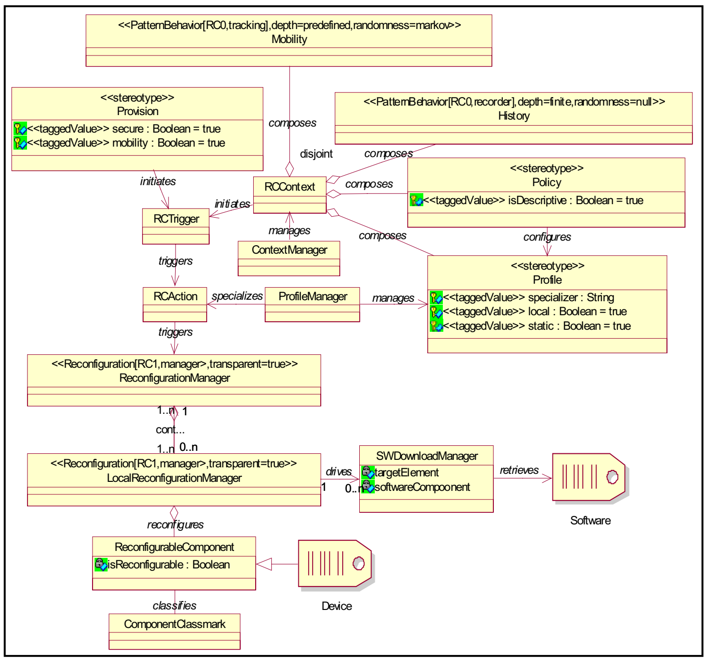
[10] IEEE Personal Communications: Special issue on Software Radios, Aug. 99, Vol. 6, No. 4.

[11] IEEE / JSAC Special issue on Software Radios, Apr. 99, Vol. 17, No. 4



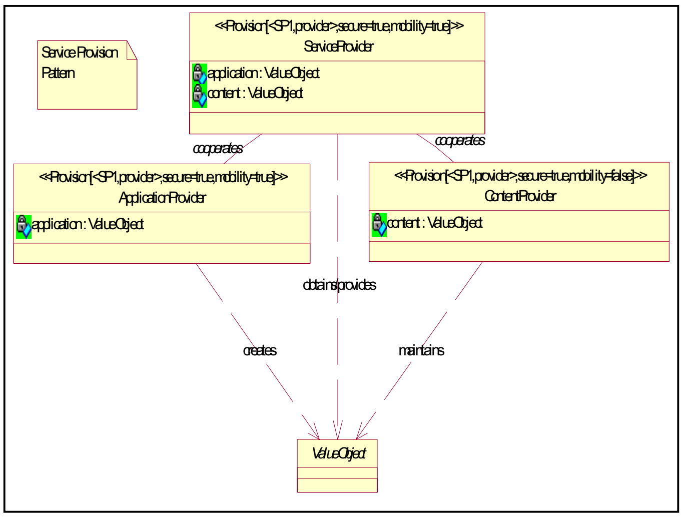
<<stereotype>> Profile
<<taggedValue>> specializer : String
<<taggedValue>> local : Boolean = true
<<taggedValue>> static : Boolean = true
<<stereotype>>
Class
<<stereotype>>
<<stereotype>> Classmark
<<taggedValue>> type : String = static
<<stereotype>> PatternClass
<<taggedValue>> pattern[0..*] : <name:string[instance:integer],role:string>
<<stereotype>> Policy
<<taggedValue>> isDescriptive : Boolean = true
<<stereotype>>
Constraint
<<stereotype>> Reconfiguration
<<taggedValue>> transparent : Boolean = true
<<stereotype>> Provision
<<taggedValue>> secure : Boolean = true
<<taggedValue>> mobility : Boolean = true
Dependency
<<stereotype>>
<<stereotype>> Reconfigurability
PatternBehavior
<<taggedValue>> depth = finite
<<taggedValue>> randomness = brownian
<<stereotype>>
Operation
<<stereotype>>
<<stereotype>> PatternOperation
<<taggedValue>> pattern[0..*] : <name:string[instance:integer],role:string>
Stereotypes' Definitions
<<stereotype>> Trigger
<<taggedValue>> description : String = intention
<<taggedValue>> type : String = event

Figure 2. Stereotype Patterns



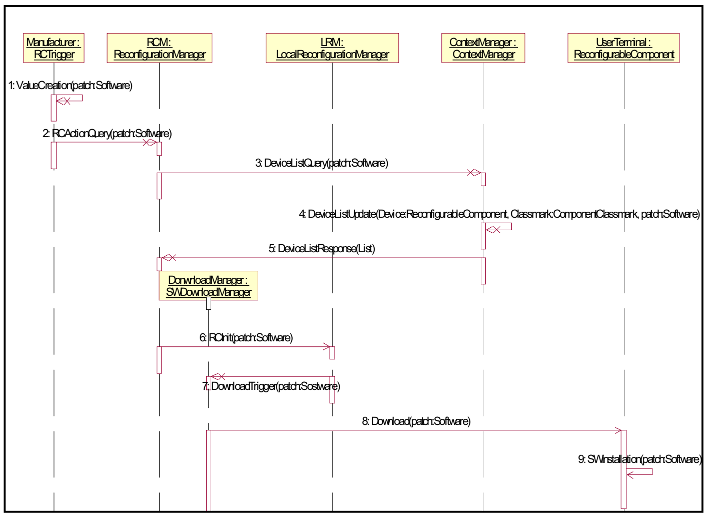
<<PatternBehavior[RC0,tracking],depth=predefined,randomness=markov>> Mobility
<<stereotype>> Provision
<<taggedValue>> secure : Boolean = true
<<taggedValue>> mobility : Boolean = true
composes
disjoint
<<PatternBehavior[RC0,recorder],depth=finite,randomness=null>> History
composes
initiates
RCContext
composes
<<stereotype>> Policy
<<taggedValue>> isDescriptive : Boolean = true
initiates
RCTrigger
manages
composes
configures
ContextManager
triggers
<<stereotype>> Profile
<<taggedValue>> specializer : String
<<taggedValue>> local : Boolean = true
<<taggedValue>> static : Boolean = true
RCAction
specializes
ProfileManager
manages
triggers
<<Reconfiguration[RC1,manager>,transparent=true>> ReconfigurationManager
1..n
1
cont...
1..n
0..n
<<Reconfiguration[RC1,manager>,transparent=true>> LocalReconfigurationManager
drives
1
0..n
SWDownloadManager
targetElement
softwareCompoonent
retrieves
Software
reconfigures
ReconfigurableComponent
isReconfigurable : Boolean
Device
classifies
ComponentClassmark

Figure 5. Reconfiguration Pattern



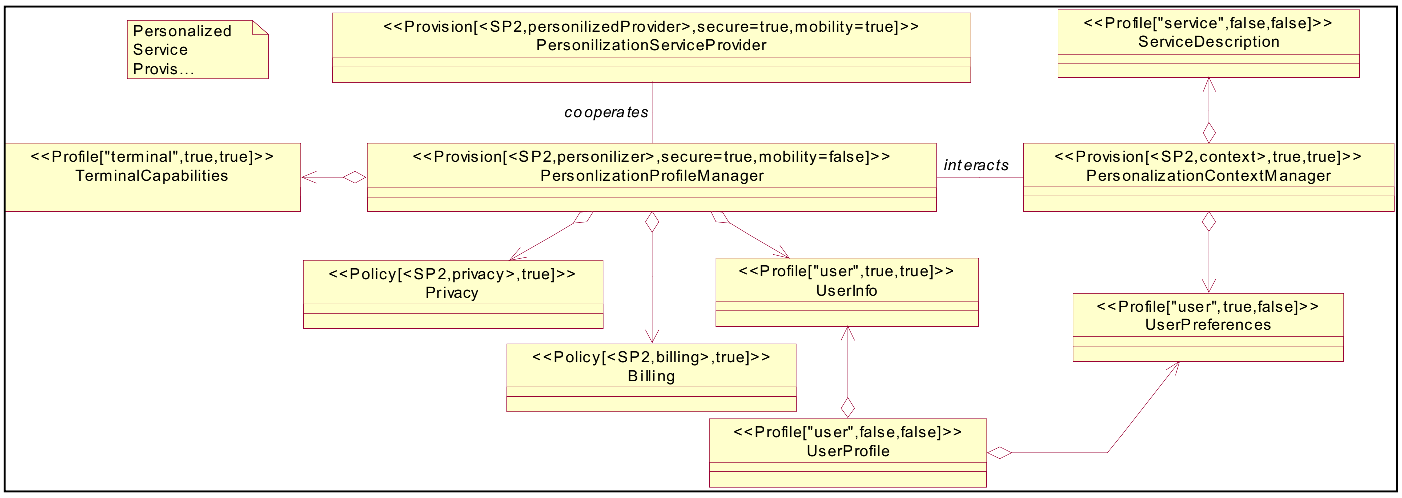
Service Provision Pattern
<<Provision[<SP1,provider>,secure=true,mobility=true]>> ServiceProvider
application : ValueObject
content : ValueObject
cooperates
cooperates
<<Provision[<SP1,provider>,secure=true,mobility=true]>> ApplicationProvider
application : ValueObject
<<Provision[<SP1,provider>,secure=true,mobility=false]>> ContentProvider
content : ValueObject
obtains/provides
creates
maintains
ValueObject

Figure 3. Value Added Service Creation Pattern



Manufacturer : RCTrigger
RCM: ReconfigurationManager
LRM: LocalReconfigurationManager
ContextManager : ContextManager
UserTerminal : ReconfigurableComponent
1: ValueCreation(patch:Software)
2: RCActionQuery(patch:Software)
3: DeviceListQuery(patch:Software)
4: DeviceListUpdate(Device:ReconfigurableComponent, Classmark:ComponentClassmark, patch:Software)
5: DeviceListResponse(List)
DonwloadManager : SWDownloadManager
6: RCInit(patch:Software)
7: DownloadTrigger(patch:Sostware)
8: Download(patch:Software)
9: SWInstallation(patch:Software)

Figure 6. Reconfiguration Use Case



Personalized Service Provis...
<<Provision[<SP2,personilizedProvider>,secure=true,mobility=true]>> PersonilizationServiceProvider
<<Profile["service",false,false]>> ServiceDescription
cooperates
<<Profile["terminal",true,true]>> TerminalCapabilities
<<Provision[<SP2,personilizer>,secure=true,mobility=false]>> PersonlizationProfileManager
interacts
<<Provision[<SP2,context>,true,true]>> PersonalizationContextManager
<<Policy[<SP2,privacy>,true]>> Privacy
<<Profile["user",true,true]>> UserInfo
<<Profile["user",true,false]>> UserPreferences
<<Policy[<SP2,billing>,true]>> Billing
<<Profile["user",false,false]>> UserProfile

Figure 4. Personalized Service Provision Pattern