SUBMISSION TO THE
SDR FORUM 2007 TECHNICAL CONFERENCE

TITLE:

CONFIGURATION CONTROL MECHANISMS IN RECONFIGURABLE RADIO EQUIPMENT

Authors:

Stoytcho Gultchev, Klaus Moessner (University of Surrey, United Kingdom)

Corresponding Author:
Stoytcho Gultchev (The University of Surrey, GU2 7XH Guilford, United Kingdom)
Tel: +44 1483 683 605, Email: s.gultchev@surrey.ac.uk

CONFIGURATION CONTROL MECHANISMS IN RECONFIGURABLE RADIO EQUIPMENT

Stoytcho Gultchev, Klaus Moessner
Mobile Communications Research
Centre for Communication Systems Research,
University of Surrey Guildford, Surrey, UK - GU2 7XH
(S.Gultchev, K.Moessner@surrey.ac.uk)

ABSTRACT

SDR technology is advancing in new stage where specification and standardization efforts are eminent. This makes it an increasingly important technology for existing as well as future wireless technologies. These efforts however must not remain limited to individual pieces of equipment, but a consistent and verified specification for the whole communication chain is required. The functions and architecture facilitating reconfiguration in E^2R [1] form a prime example. To be able to guarantee interoperability and cooperation between the different elements within such system, a detailed description of the requirements for equipment but also of the control operations that form and enable a reconfiguration process is needed. These specifications need to include descriptions of the actual module functionality as well as of the E^2R management/ control sequences. The main requirements are towards compliance as well as towards software and (radio) configuration verification and validation techniques. This paper provides an overview of the requirements and describes examples of how interoperability in End-to-End Reconfigurable Systems can be achieved.

1. INTRODUCTION

In the E^2R approach, the definition and implementation of mechanisms that control reconfigurable hardware is of high importance; this includes the CCM (configuration control module) and its links to the HAL (hardware abstraction layer). The flow of control messages between the different modules (i.e. CMM (configuration management module), CCM and CEM (configuration execution module)), as well as those between the CCM sub-modules, have been defined. The interfaces as well as state machines of the modules have been developed, and means for verification of procedures as well as of the functionality for each module have been investigated using different test cases.
This paper addresses and documents the CCM specification and its verification. It provides a coherent representation of the control interfaces of the CCM and its role and impact in a reconfiguration management architecture. The interface extensions discussed complement the overall architectural framework for E^2R terminals, see Figure 1. Part of this framework is the differentiation between reconfiguration support and reconfigurable radio functionality, the model was evaluated using formal methods as well as a software demonstrator.
The paper provides a detailed description of the CCM operations, and the verification method used. It also describes the demonstrator developed and the implementation of different use cases.
The paper also describes the design approach using UML for the overall system and functionality definition, and SDL for the design of the information flow and verification mechanisms of the system is used [2]. It describes the design approach as well as the means and mechanisms defined for the verification of radio configurations.

2. OPERATIONS AND CONTROL PROCESSES IN E^2R SYSTEMS

One of the cabailities expected from SDR terminals (in the commercial domain) is that they are able to reconfigure between different air interfaces. A reliable way to manage and control this “vertical” reconfiguration of a mobile node is required. Only if such management/control, transparent to the user, is achieved, SDR technology will be viable and usable in the commercial arena.

Reconfiguration controllers need to be aware of the configurations resource requirements as well as the capabilities of the existing (hardware and software) platform. This has been investigated in the End-to-End Reconfigurability (E^2R) project [3], where the definition of management entities follows a modularised approach used in the overall system architecture specification of E^2R. The modules include:

- ✓ Configuration Management Module (CMM): functional entity implementing high level configuration management, CMM handles the network reconfiguration requirements and the instantiation of configurations on the hardware platform;

- ✓ Configuration Control Module (CCM): functional entity facilitating the control of reconfigurable hardware and software. The CCM implements different reconfiguration scenarios and handles resource scheduling on the implementation platform;
- ✓ Configuration Execution Module (CEM): reconfigurable hardware module (i.e. part of the reconfigurable SDR platform) implementing algorithms such as modulator, interleaver, etc.

Based on the E^{2}R reconfiguration control architecture and a set of reconfiguration scenarios, the design and functionality of the system, as well as the mechanisms to control the reconfigurable hardware are provided. In addition, the flow of control messages between the different modules (i.e. CMM, CCM and CEM) concentrating on the control signals between the CCM sub-modules is presented.

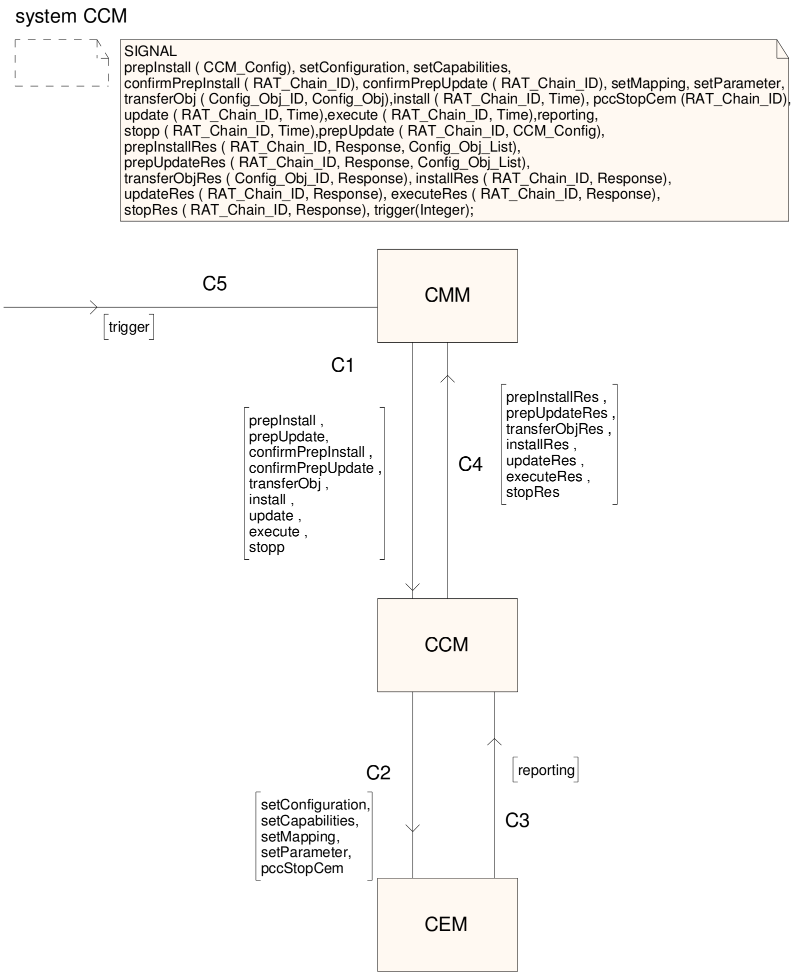
```
SIGNAL
prepInstall ( CCM_Config), setConfiguration, setCapabilities,
confirmPrepInstall ( RAT_Chain_ID), confirmPrepUpdate ( RAT_Chain_ID), setMapping, setParameter,
transferObj ( Config_Obj_ID, Config_Obj),install ( RAT_Chain_ID, Time), pccStopCem (RAT_Chain_ID),
update ( RAT_Chain_ID, Time),execute ( RAT_Chain_ID, Time),reporting,
stopp ( RAT_Chain_ID, Time),prepUpdate ( RAT_Chain_ID, CCM_Config),
prepInstallRes ( RAT_Chain_ID, Response, Config_Obj_List),
prepUpdateRes ( RAT_Chain_ID, Response, Config_Obj_List),
transferObjRes ( Config_Obj_ID, Response), installRes ( RAT_Chain_ID, Response),
updateRes ( RAT_Chain_ID, Response), executeRes ( RAT_Chain_ID, Response),
stopRes ( RAT_Chain_ID, Response), trigger(Integer);
```
system CCM
C5
trigger
CMM
C1
prepInstall , prepUpdate, confirmPrepInstall , confirmPrepUpdate , transferObj , install , update , execute , stopp
C4
prepInstallRes , prepUpdateRes , transferObjRes , installRes , updateRes , executeRes , stopRes
CCM
C2
setConfiguration, setCapabilities, setMapping, setParameter, pccStopCem
reporting
C3
CEM

Figure 1 System CCM block diagram

3. CONTROL MECHANISMS AND SCENARIOS

The overall architecture specified as System CCM and shown in Figure 1 is a combined functional description (in SDL) of the CCM, CMM and CEM modules.

The **CMM** is responsible for managing all configuration tasks in the equipment. It is depicted in Figure 2 showing the internal architecture and signaling processes between different CMM modules. The CMM contains following entities or sub-modules:
The '*Negotiation and Selection*' (CMM_NS) module manages the negotiation between with the various available networks. The selection procedure takes into account information, such as user and equipment profile and the service offers of the available networks.

The '*Configuration Download*' (CMM_Dwnld) module provides the capability to perform downloads of the different software components used by the reconfiguration process. The download procedure encompasses mechanisms and signaling to perform negotiation with the network side for software downloads. The download procedure can be network or equipment initiated.

The '*Configuration Profiles*' (CMM_Prof) module provides configuration profiles, information on applications, user classes, equipment classes/capabilities and configuration data models. It manages the profile repository and retrieves profiles on request from other modules.

The '*Decision-Making and Policy Enforcement*' (CMM_DMP) module is a module that mainly supports context and policy management procedures. It is responsible for the coordination of the reconfiguration implementation. During the process of software download this module chooses the appropriate software running using a set of decision algorithm and applies them on the information retrieved from the CMM_Prof. Moreover, the CMM_DMP provides to CMM_NS with policies for a given RAT.

The '*Reconfiguration Installation*' (CMM_Inst) module provides the means for configuration representation, reliable configuration deployment and configuration rollback, which involves installation and switching, and non-volatile logging.

The '*Configuration Event Handler*' (CMM_Evnt) module provides a set of interfaces to the CCM. Via these interfaces the CMM_Evnt receives messages and trigger events from the CCMs and dispatches them to the appropriate sub-module of the CMM for processing. All communication, independent of the direction, between CMM sub-modules and the different CCMs is processed via the CMM_Evnt.

The **CCM** initiates, coordinates and performs the different reconfiguration functions, as shown in Figure 3. It communicates with the CMM via the *CCM_Event Interfaces* as shown in Figure 1.

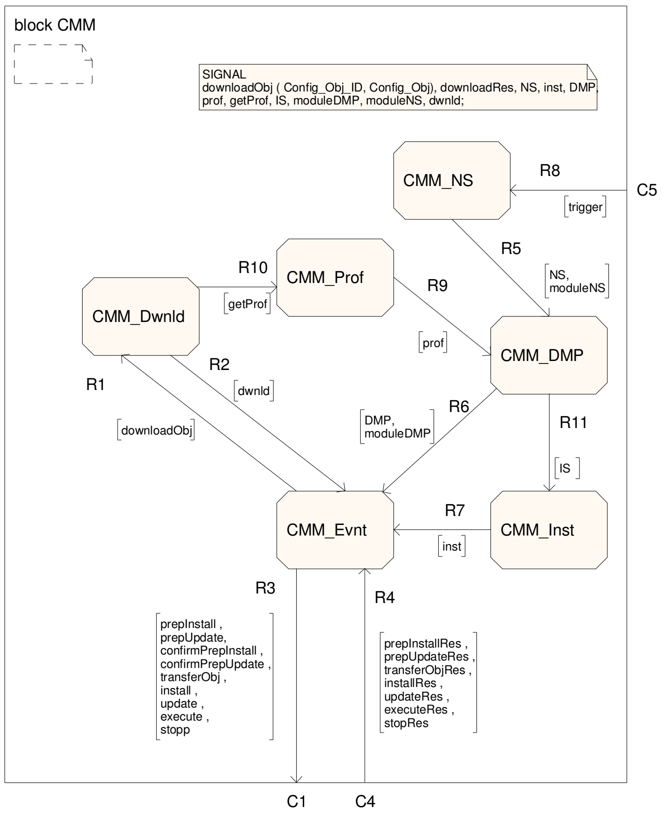
block CMM
SIGNAL
downloadObj (Config_Obj_ID, Config_Obj), downloadRes, NS, inst, DMP, prof, getProf, IS, moduleDMP, moduleNS, dwnld;
CMM_NS
R8
C5
trigger
R5
NS, moduleNS
R10
CMM_Prof
R9
CMM_Dwnld
getProf
prof
CMM_DMP
R2
R1
dwnld
R6
R11
downloadObj
DMP, moduleDMP
IS
R7
CMM_Evnt
CMM_Inst
inst
R3
R4
prepInstall , prepUpdate, confirmPrepInstall , confirmPrepUpdate , transferObj , install , update , execute , stopp
prepInstallRes , prepUpdateRes , transferObjRes , installRes , updateRes , executeRes , stopRes
C1
C4

Figure 2 CMM block diagram

The *CCMServiceManager* dispatches incoming CMM_Evnt messages to the internal CCM_RM functional entities and vice a versa.

Primary Configuration Control (PCC) Engine – plans the actual new configuration; it translates the overall configuration into a set of sub-configurations for the different layers (application, protocol and physical layer). The PCC is also used to delete non-required functional elements from the current instantiated RAT, to free resources for the new configuration;

Secondary Configuration Control (SCC) Engine - supports all activities related to a currently instantiated RAT (i.e. supports CMM_NS);

Capability Server determines the configurations the CEMs are able to support;

CCMDatabase - contains configuration objects, i.e. binary files, cached in the CCM and downloaded into the CEMs.

Going further down into the mechanisms closer to the radio implementation platform, the CCM_RM handles a set of resources called **Configurable Execution Modules**. CEMs control the resources at the harware layer, e.g. DSPs and reconfigurable logic. The CCM configures (via the CEM "Service and Reporting interface") each of these resources and defines the operating environment for each of them (i.e. allocates memory use, etc). Due to its specific manufactures implementation this module is not implemented in full and only a abstract representation is used in the definition.

EVALUATION SCENARIOS

A range of scenarious have been used to demonstrate the validity of the functional description of the system. However, there are two basic scenarios that cover almost all of the functions required, these two have been considered and used for the evaluation of the architecture. The "***boot-up***" and "***system reconfiguration***" are used to illustrate the control and management functions on the different system levels (i.e. from top level decision making to instantiation of configurations on the radio implementation platform)

The "Boot-Up" Scenario

This scenario represents an initial boot-up sequence of an E^2R terminal with an initial configuration available. It assumes the trigger to start implementation of a particular configuration is external (i.e. power on).

In this case objects are transfered to the CCMDatabase and the terminal capability profile within the CCM Capability Server is updated.

Once the CMM receives the confirmation message from the Capability Server, it sends a message to the CCM to install the requested configuration within the pre-defined duration, i.e. meeting the 'useful installation' deadline. The CCMServiceManager passes the message to the PCC Engine, to process the install command. Then, the PCC Engine gets the required configuration file ("black box code") from the CCM Database via the CCMServiceManager and configures the CEM according to the configuration information (i.e. the PCC does not have to understand the type or functioning of the code it installs, it merely has to implement the required installation steps).

Once the configuration process of the CEM starts, it will perform four distinct steps:

1) Download configuration black box code from the CCM to the CEM.
2) Label the functional capabilities of the black box code.
3) Provide Mapping of the black box code to the physical environment.
4) Parameterize the algorithms independently of the physical implementation.

Once the CEM sends its confirmation about successful configuration, the PCC Engine sends a message (dbSetupObjReq) to CCMDatabase to upload the relevant objects to the CEM (i.e. updating the terminal profile). When the PCC Engine receives the confirmation for the completion of the upload, it sets parameters onto the CEM and confirms to the CMM the success of the installation procedure. Once this is finished, the CMM sends the execution command to the PCC Engine in order to activate the installed configuration in the CEM. The PCC Engine responds back to the CMM with a message confirming the completion of the configuration.

The "System Reconfiguration" Scenario

This second scenario may occur in different occasions:

- A module in the working radio chain does not conform to its design specifications.
- The base-station has detected incorrect behaviour of the terminal.
- The software or terminal supplier provides a module upgrade.

The sequence of this scenario represents a module update in an operating terminal. The reconfiguration again starts as the CMM sends to CCMServiceManager to prepare to update a module in the already implementated chain. The CCMServiceManager requests from the Capability Server to check the availability of the required update.

Again, if any of the update required objects is missing, the CMM becomes informed and can then transfer these missing objects to the CCM Database. The CCMServiceManager updates the profile within the CCM Capability Server. As a next step, the CMM sends a (confirmPrepUpdate) message to check the readiness of the CCM Capability Server and a confirmation is sent back to the CMM. After that, the CMM sends an instruction to PCC Engine to stop the functioning of the CEM within the pre-defined duration, i.e. deadline. After stopping the CEM, the CMM issues a message to the PCC Engine to update the current configuration within the pre-defined time duration, i.e. deadline. In the next sequence the PCC Engine requires from CCM Database the configuration file and uses it to configure CEM. It also sets the functional capabilities information and physical mapping information to the CEM according to the configuration information.

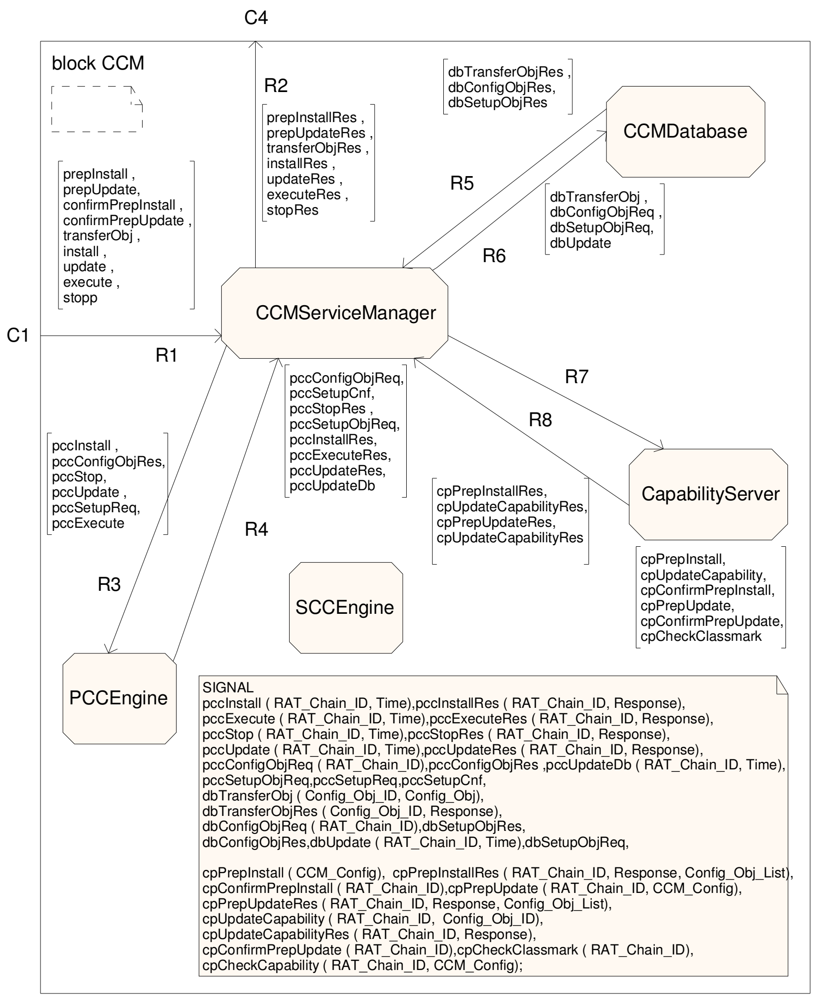
C4
block CCM
R2
prepInstallRes , prepUpdateRes , transferObjRes , installRes , updateRes , executeRes , stopRes
dbTransferObjRes , dbConfigObjRes, dbSetupObjRes
CCMDatabase
prepInstall , prepUpdate, confirmPrepInstall , confirmPrepUpdate , transferObj , install , update , execute , stopp
R5
dbTransferObj , dbConfigObjReq , dbSetupObjReq, dbUpdate
R6
CCMServiceManager
C1
R1
pccConfigObjReq, pccSetupCnf, pccStopRes , pccSetupObjReq, pccInstallRes, pccExecuteRes, pccUpdateRes, pccUpdateDb
R7
R8
pccInstall , pccConfigObjRes, pccStop, pccUpdate , pccSetupReq, pccExecute
cpPrepInstallRes, cpUpdateCapabilityRes, cpPrepUpdateRes, cpUpdateCapabilityRes
CapabilityServer
R4
cpPrepInstall, cpUpdateCapability, cpConfirmPrepInstall, cpPrepUpdate, cpConfirmPrepUpdate, cpCheckClassmark
R3
SCCEngine
PCCEngine
SIGNAL
pccInstall (RAT_Chain_ID, Time),pccInstallRes (RAT_Chain_ID, Response),
pccExecute (RAT_Chain_ID, Time),pccExecuteRes (RAT_Chain_ID, Response),
pccStop (RAT_Chain_ID, Time),pccStopRes (RAT_Chain_ID, Response),
pccUpdate (RAT_Chain_ID, Time),pccUpdateRes (RAT_Chain_ID, Response),
pccConfigObjReq (RAT_Chain_ID),pccConfigObjRes ,pccUpdateDb (RAT_Chain_ID, Time),
pccSetupObjReq,pccSetupReq,pccSetupCnf,
dbTransferObj (Config_Obj_ID, Config_Obj),
dbTransferObjRes (Config_Obj_ID, Response),
dbConfigObjReq (RAT_Chain_ID),dbSetupObjRes,
dbConfigObjRes,dbUpdate (RAT_Chain_ID, Time),dbSetupObjReq,
cpPrepInstall (CCM_Config), cpPrepInstallRes (RAT_Chain_ID, Response, Config_Obj_List),
cpConfirmPrepInstall (RAT_Chain_ID),cpPrepUpdate (RAT_Chain_ID, CCM_Config),
cpPrepUpdateRes (RAT_Chain_ID, Response, Config_Obj_List),
cpUpdateCapability (RAT_Chain_ID, Config_Obj_ID),
cpUpdateCapabilityRes (RAT_Chain_ID, Response),
cpConfirmPrepUpdate (RAT_Chain_ID),cpCheckClassmark (RAT_Chain_ID),
cpCheckCapability (RAT_Chain_ID, CCM_Config);

Figure 3 CCM block diagram

4. RECONFIGURATION VERIFICATION

Boot_up and system reconfiguration scenarious provide the basic procedures to required to ensure reliable connectivity for reconfigurable terminals.

However, the complexity of configurations cannot be just classified in a simple specification, as research in E^2R has shown [4]. It will require much more to reflect the whole reconfiguration validation process as investigated in [2]. A variety of factors influence the validation processes for radio configurations, the most important of which are portability, interoperability, spectrum efficiency and compatibility between platforms and reconfigurable systems. The result is a common reconfigurable framework enabling standard compliant implementations of radio configurations. .

To achieve development of a system where software configurations implement fully compliant radio nodes, a systematic approach for every area mentioned above with consensus between reconfiguration service and equipment providers on a generic overall reconfigurable architecture is needed. The approach as described in [2] has been applied to verify the above (use)cases and validate the supplied framework.

After complete modeling of the scenarios, they have been simulated and validated with the object communication protocol.

In this work, state space exploration and state transition diagrams were used for validation of the design specification.

State transition diagrams were used for the analysis of the system modules, checking the states and the signals for whether they show the desired results. For the purpose of simplicity and page restrictions the details of all the modules' state transition diagram from the SDL Validator, applied to the System CCM which show the states of the modules, and what signals trigger the modules to action and the current state changes to another state are omitted from this paper.

The SDL specification for the System CCM model has been revised following the results from simulation and validation. The specification has been tested for completeness as well as for deadlocks and starvation points. The symbol and transition coverage of the specification are both 100%, which means the state space of the system is reachable and completely explorable.

The SDL implementation formally specifies the function design of the modules in the System CCM, and the communication sequences between the modules. With the SDL specification, the E^2R management and control architecture is formally modeled by executable specification, which can be simulated and validated before the architecture is implemented and coded darkly. Through the SDL specification, the architecture is verified and validated in an early stage, which is valuable for complex system design and save the cost of software development.

In practice testing and validation are closely related, and after the SDL system is completely validated, there is no further testing required. The scenarios have been tested and shown in the simulation MSCs (contained in a number of E^2R deliverables). This also includes validation of the correctness and design of the mechanisms between CMM and CCM.

5. SUMMARY

SDR technologies will be a major part of any future adaptive communication platform. The paper presented some of the important factors for reconfiguration verification and different verification techniques that need to be deployed for the uninterrupted and correct functioning of E^2R management and control modules. The different verification methods have been highlighted in validation of reconfiguration systems and their reconfiguration management and control parts.

Finally, an example of the implementation and verification of reconfigurable radios and complement the reconfiguration plane concept that was introduced with the System CCM framework has been provided.

6. ACKNOWLEDGMENTS

This work was performed in project E^2R II which has received research funding from the Community's Sixth Framework programme. This paper reflects only the authors' views and the Community is not liable for any use that may be made of the information contained therein. The contributions of colleagues from E^2R II consortium are hereby acknowledged.

7. REFERENCES

[1] IST project E^2R, https://e2r2.motlabs.com/

[2] Gultchev S, “Verification of software reconfiguration platforms” SDR05 Technical Conference, Orange County, California, USA , 14-18 November, 2005

[3] Gultchev S, et.al. “Control Mechanism to Enable Equipment Reconfiguration and Facilitate Operation of E2R Systems” SDR05 Technical Conference, Orange County, California, USA, 14-18 November, 2005

[4] Berzosa F, "Reconfiguration Terminal Classmarks", WWRF 14, San Diego, California, USA, 07-08 July 2005.