Cognitive Radio Experimentation World

Project Deliverable D5.2
Initial report on demand-driven federation functionality

Contractual date of delivery:	30-09-2012
Actual date of delivery:	30-09-2012
Beneficiaries:	IBBT, IMEC, TCD, TUB, TUD, TCS, EADS, JSI
Lead beneficiary:	TUB
Authors:	Danny Finn (TCD), Justin Tallon (TCD), Luiz DaSilva (TCD), Michael Mehari (IBBT), Wei Liu (IBBT), Stefan Bouckaert (IBBT), Ingrid Moerman (IBBT), Christoph Heller (EADS), Somsaï Thao (TCS), Alejandro Sanchez (TCS), David Depierre (TCS), Peter Van Wesemael (imec), Mattias Desmet (imec), Jan Hauer (TUB), Mikolaj Chwalisz (TUB), Nicola Michailow (TUD), Zoltan Padrah (JSI)
Reviewers:	IBBT/TUD
Workpackage:	WP5 – Demand Driven Extensions
Estimated person months:	25
Nature:	R
Dissemination level:	PU
Version	1.0

Abstract: This document describes the initial demand-driven extensions of the CREW federation. The extensions are an instrument to upgrade the CREW federation with a new set functionality beyond the basic functionality developed in other CREW work packages. On the one hand they have been derived in a demand-driven and open way from feedback by CREW core members and their involvement in the FIRE community – this includes the adoption (and refinement) of the Connectivity Brokerage framework, the joint development of a common cognitive radio language together with the GENI initiative and hardware related extensions. On the other hand, extensions and supporting actions have been a result of the new partners who have joined the CREW project via the first open call (WP7).

Keywords:

cognitive radio, wireless networks, testbed, spectrum sensing, context awareness

Executive Summary

This document summarizes the initial demand-driven extensions of the CREW federation. It describes how the federated CREW test facilities have been extended with a first set of new functionality beyond the basic functionality developed in WP3 and WP4. This functionality has been defined in a demand-driven and open way based on the gaps identified from (1) feedback on experimentation by CREW core members and their involvement in the FIRE community as well as (2) external experimenters who have joined the project as new partners (WP7).

The corresponding CREW Task 5.1 "New functionality" addresses the extension of the CREW federation and the initial report on these demand-driven extensions is given in this document - the final report on demand-driven extensions (and FIRE support actions) will be reported in D5.3 [M36].[1]

One extension of the CREW platform is the adoption (and refinement) of the *Connectivity Brokerage* framework, an architecture that enables coordination and cooperation between heterogeneous networking components, developed at UC Berkeley. This framework now enables CREW experimenters to integrate their mechanisms-under-test in a specified CR environment and facilitates experimentation by enabling reuse of CR software components and concepts. In addition, the CREW involvement in the FIRE community resulted in collaboration between CREW and the GENI initiative, which manifested in the joint development of a common cognitive radio language.

Furthermore, integration of the following new hardware was identified as a valuable extension to the CREW federation: integration of low-cost USB spectrum sensing devices into the TWIST testbed, extension of the TCD testbed with 10 additional nodes and extension of the LTE testbed of TUD with with 2.1 GHz front-ends. Task 5.1 also covered the necessary actions to support the experiments carried out by the new project partners who participated in CREW WP7 via the first open call. A description of these support actions is also reported in this document.

[1] Note that this document (D5.2) does not cover the FIRE support actions (other than related to demand driven extensions), because FIRE support actions are described in D5.1 and D5.3 only.

List of Acronyms and Abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ADC	Analog to Digital Converter
AGRAC	Automatic Gain and Resource Activity Controller
AI	Air Interface
API	Application Programming Interface
ASIP	Application-Specific Instruction-set Processor
ATSC	Advanced Television Systems Committee
AWGN	Additive White Gaussian Noise
BAN	Body Area Network
BWRC	Berkeley Wireless Research Center
BS	Base Station
BTS	Base Transceiver Station
CAgent	Connectivity Agent
CB	Connectivity Brokerage
CBAN	Cognitive Body Area Network
CIC	Cascaded Integrator-Comb
CoAP	Constrained Application Protocol
CompNet	Composite Network Agent
CORDIC	COordinate Rotation DIgital Computer
COTS	Commercial Off-The-Shelf
CP	Cyclic Prefix
CR	Cognitive Radio
CREW	Cognitive Radio Experimentation World
CSMA	Carrier Sense Multiple Access
DC	Direct Current
DDR RAM	Double-Data-Rate Synchronous Dynamic Random Access Memory
DFE	Digital FrontEnd
DIFFS	DIgital Front end For Sensing
DVB-T	Digital Video Broadcasting - Terrestrial
EVA	Extended Vehicular A
FCC	Federal Communications Commission
FDD	Frequency Division Duplex
FER	Frame Error Ratio
FFT	Fast Fourier Transformation
FIR	Finite Impulse Response
FIRE	Future Internet Research and Experimentation Initiative
FPGA	Field Programmable Gate Array
GENI	Global Environment for Network Innovations
GPS	Global Positioning System
GUI	Graphical User Interface
HID	Human Interface Device
ID	Identifier
IO	Input/Output

IPC	Interprocess Communication
IRIS	Implementing Radio In Software
I/Q	In-Phase / Quadrature-Phase
ISM	Industrial Scientific Medical
LAN	Local Area Network
LTE	Long Term Evolution
MAC	Medium Access Control
NFS	Network File System
NTP	Network Time Protocol
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
OS	Operating System
PA	Platform Agent
PC	Personal Computer
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PD	Probability of Detection
PFA	Probability of False Alarm
PLL	Phase Locked Loop
PMD	Probability of Missed Detection
PRB	Physical Resource Block
PSD	Power Spectral Density
PUB	Publish
RF	Radio Frequency
RFIC	Radio Frequency Integrated Circuit
ROC	Receiver Operating Characteristics
RSSI	Received Signal Strength Indication
RTT	Round Trip Time
Rx	Receiver
SDR	Software Defined Radio
SFTP	Secure FTP
SIMD	Single Instruction, Multiple Data
SIR	Signal Interference Ratio
SNR	Signal to Noise Ratio
SUB	Subscribe
SUT	System under Test
TCP	Transmission Control Protocol
TDD	Time Division Duplex
TSMC	Taiwan Semiconductor Manufacturing Company
TVWS	Television White Spaces
TWIST	TKN Wireless Indoor Sensor network Testbed
Tx	Transmitter
UDP	User Datagram Protocol
UML	Unified Modeling Language
UniNet	Unified Network Agent
US	Usage Scenario (see D2.1 for definition)

USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
VCC	Virtual Control Channel
WARP	Wireless Open Access Research Platform
WLAN	Wireless Local Area Network
WSAP	White Space Access Point
WSD	White Space Device
WSN	Wireless Sensor Network
ZMQ	ZeroMQ

Table of Contents

1 Introduction

In the CREW project demand-driven extensions are the tool to dynamically adapt the CREW federation to needs identified during the course of the project. While some of these extensions were envisioned during the project planning, not all needs were evident at that time. In addition, since external experimenters were joining the project as a result of open calls during the project course (M12/M24) a mechanism to extend the project with new functionality to better support these experimenters was required.

The first set of extensions were identified based on the CREW internal use cases: during the first months of the project it became apparent that the CREW federation was lacking a framework that allows experimenters to abstract from certain cognitive radio software components to better concentrate only on the software components of interest. For example, an experimenter who investigates a sensing algorithm may not be interested in how/when data is communicated to the database or how the database is implemented but instead use predefined software components and interfaces for these tasks. To this end the *Connectivity Brokerage* framework, an architecture that enables coordination and cooperation between heterogeneous networking components developed at UC Berkeley, was adopted and extended for the CREW project (Section 2.1).

In addition, the CREW involvement in the FIRE community resulted in collaboration between CREW and the GENI initiative. One result envisaged in this collaboration is to realize a common API (common cognitive radio language) for enabling experiments on top of the CREW cognitive radio devices. This extension is described in Section 2.2.

Furthermore, a set of new hardware was identified as a valuable addition to the CREW federation. The selected hardware extends the CREW functionality to enable new or more complex (large-scale) experiments and thus increases the attractivity of the CREW federation. The new hardware (and the additional software suppport) are described in Section 2.3.

The last set of demand-driven extensions was derived from external experiments carried out by the new project partners who joint the CREW project as a result of the first open call (WP7). The resulting extensios and support actions are described in Section 3; finally, this document is summarized and concluded in Section 4.

2 Demand-driven Extensions Derived From Internal Use Cases

This section describes the extensions that were developed as a result of the shortcomings identified in the CREW internal use cases.

2.1 Connectivity Brokerage Framework

The CREW project infrastructure includes a number of different hardware modules for cognitive radio research, ranging from low cost commercial-of-the-shelf (COTS) units - such as WiSpy USB spectrum analyzers[2] - to more advance sensing solutions like the IRIS platform or IMEC sensing agent. An experimenter using the CREW facilities may want to involve multiple heterogeneous modules, e.g. combine spectrum sensing information from different sources; furthermore, an experimenter may want to abstract from those CR functionalities that are not of primary interest to the investigation. For example, an experimenter may want to build a database of spectrum sensing data obtained from WiSpy modules and IMEC sensing agents to see if the combination can improve spectral sensing performance; but the experimenter may not be interested in how/when data is communicated to the database or how the database is implemented.

The *Connectivity Brokerage* (CB) framework [1], developed at UC Berkeley, allows experimenters to focus their investigations only on the subset of the cognitive radio approaches of interest and establishes a means for collaboration among different CR entities at different levels of abstraction. The benefits for the CREW project and potential experimenters are that the CB concept clearly structures and defines functionality in a CR system. Consequently, experimenters can better focus on / distillate their individual tasks. In addition, experimenters may use the CB framework to "fill up" the missing parts of their CR system-under-test (SUT) with generic CB framework components.

In this section we will first explain the concept of the CB framework; we will then identify gaps and challenges we faced in adapting the framework in the CREW project; afterwards we will describe our solutions and the main design decisions we made to realize the framework in the CREW project; and finally we will describe our implementation and explain in which way the CB framework was utilized in the CREW usage scenarios (WP6).

During Year 2 of the CREW project **a collaboration has been established between CREW and UC Berkeley** through various bilateral discussions between CREW and Jan Rabaey from Berkeley Wireless Research Center (BWRC). BWRC has pioneered the Connectivity Brokerage framework and is interested in cooperating with the CREW towards prototyping the framework (see support letter in Appendix 1).

2.1.1 Concept

As explained in [1] the main goal of Connectivity Brokerage is to provide a general framework that enables diverse wireless technologies to exchange information and collaborate in a seamless fashion. This will make joint optimization of the spectrum resources possible.

The strict object-oriented strategy with clear semantics is used to accomplish CB goals. The basic components of the CB architecture are the Connectivity Agents.

2.1.1.1 Connectivity Agent (in short CAgent)

CAgent is a generic object that represents a particular interest in the brokerage arena (to stay within the same terminology). It may represent the interests of a terminal (or a user), a wireless network, or a cluster of collaborating networks.

Each CAgent, independent of its nature, is characterized by a set of parameters, has a known state (divided into public and private sections), and supports a set of common functions.

[2] http://www.metageek.net/products/wi-spy/

There exist the following CAgent classes (cf. Figure 1):

Air Interface (AI): abstracts given wirelesses interface, and provides the needed interface and control knobs for it to work properly with other CAgents.

Platform Agent (PA): represents the interests of a given (mobile/non-mobile) platform within a CB controlled space. PA CAgent policies and behavior are strongly influenced by the preferences and privileges of the platform owner (such as the networks that he/she is allowed to connect).

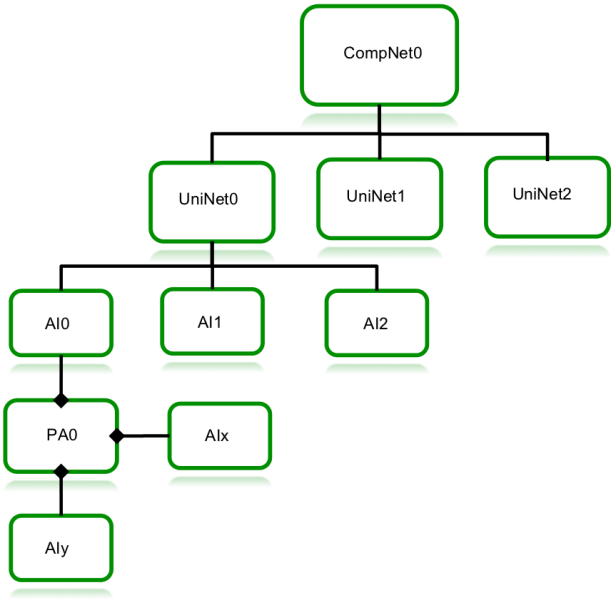

Figure 1: Example CAgent (Rabaey, et al., 2010)

Unified Network Agent (UniNet): represents a collection of AIs that adhere to a unified set of wireless communication rules or protocols, and that jointly pursue a unified optimization strategy. The most typical scenario for this would be a homogeneous network of similar AIs running under a common administration.

Composite Network Agent (CompNet): represents a collection of UnitNets that are interested in cooperation or collaboration. In essence, it is the CompNet CAgent that truly enables the concept of connectivity brokerage. By leveraging a high-level understanding of connectivity resources and needs over multiple networks, the CompNet enables solutions that exceed what the UniNets on their own can accomplish. In addition, CompNets enable and support hierarchical structures and therefore improve the scalability of the system. In its simplest nature, a CompNet just shares information between its component networks; in its most complex form, it balances the diverse requirements of the networks and trades off between diverging cost functions and metrics, while adhering to dynamically changing policy rules.

CAgents may contain the following functionalities (cf. Figure 2):

Repository – An essential requirement for effective cooperation and collaboration is that CAgents make available (at least partially) to all interested parties information learned, decisions made, and current state. These parties may be components within the CAgent, or any other CAgent that interacts with it. This is enabled through a distributed repository structure. Each CAgent will publish part of its own repository to the global CB repository, making it public. The distributed global repository constructed in this manner is the backbone of the CB framework and the fundamental component that enables cooperation and dynamic information exchange among wireless technologies.

Discovery – A cornerstone of the CB concept is the capability to actively learn the properties of the environment. The discovery function extracts and filters useful information from a massive amount of data, which can be collected from different levels and layers of the wireless systems. The resulting information is posted in the repository.

Optimization – The obtained information can be used to optimally configure the system so that performance goals or other criteria are met within the boundaries of the guiding rules and policies. The required optimizations are often implemented in a distributed or semi-distributed fashion and therefore each CAgent should be able to engage in a distributed optimization strategy.

Execution – The outcomes of the optimization process need to be conveyed to the interested parties and executed in a reliable and equitable fashion. Execution of decisions over distributed systems might require relatively complex coordinated transactions. This is why this is a fundamental function for every Connectivity Agent.

Access control – How information is gathered and disseminated through the system and who is allowed to actively participate in the automation and management process is subject to rules and trust mechanisms. Cooperative systems can become a prime target of malicious attackers and the security and access control aspects need to be explicitly addressed. Authentication of the different CAgents as well as repository data access control and association processes are part of this function.

Policy Support – Policies set the boundaries and ground rules of the optimization processes. In contrast to current practice (in which the policy is cast in stone in advance), the policies may vary dynamically, and the networks should be able to adapt these variations. This is a truly innovative feature introduced by the CB concept. It is an essential element of the "brokerage" model.

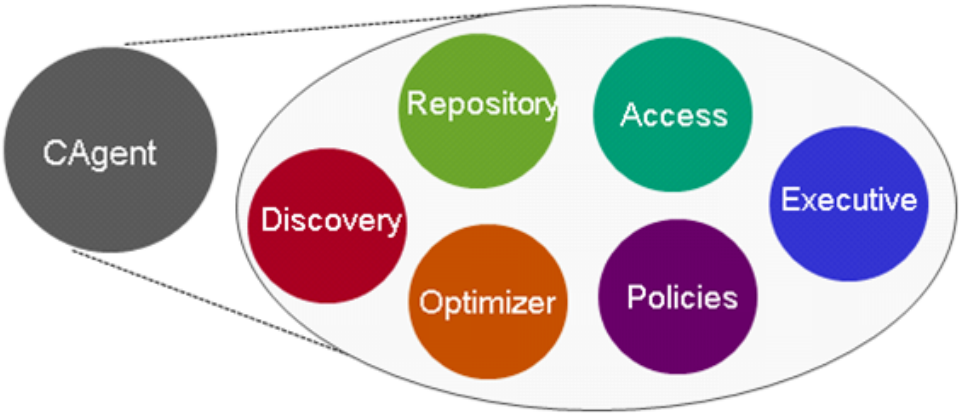

Figure 2: CAgent functionalities (Rabaey, et al., 2010)

2.1.1.2 Inter-CAgent Communication

The communication between CAgents is performed over the so-called *Virtual Control Channel* (VCC). The CB framework does not specify the details of the VCC, rather it is treated as some abstract communication channel over which CAgents typically communicate based on the publish-subscribe interaction scheme. For example, a spectrum sensing device could publish its sensing data and any number of subscribers could subscribe for the data. Since the VCC is not defined in detail, one challenge lies in the specification of the VCC, and the possible interaction patterns. This is detailed in the following chapter.

2.1.2 Connectivity Brokerage in the CREW Project

With the CB framework CREW experimenters can better structure and focus on their individual tasks. In addition, experimenters may use the CB framework to "fill up" the missing parts of their CR system-under-test (SUT) with generic CB framework components. However, to reach this goal the CREW project has to address a set of challenges: the CB framework exists mostly on a conceptual level, and while the CB framework defines the architectural components and their functionalities in detail, the interfaces between components are not specified. This lack of specification refers to (1) the interface / communication channel between different CAgents as well as (2) the interfaces among the functionalities within a CAgent (discovery, repository, optimizer, etc.). A CREW instantiation of the framework thus involved specification of the inter-CAgent as well as intra-CAgent communication in detail. In the following we list a set of communication requirements and present a specification of the communication interfaces used within the CREW project.

2.1.2.1 Inter-CAgent Communication (VCC Specification)

The CB framework defines an abstraction of communication backplane, called the Virtual Control Channel (or VCC), which can be used to exchange messages among CAgent. According to the CB architecture CAgents communicate with each other over the VCC using a uniform interface protocol, however, the architecture leaves the actual implementation of the communication protocol undefined. We have identified the following requirements that a VCC must meet in the CREW project:

- Enable message exchange among CAgents located on different hosts. We assume that these hosts are able to connect via the TCP/IP protocol (other types of connectivity, such as CoAP/UDP/6LowPAN for constrained embedded sensor devices may be considered in future extensions / CREW Year 3).
- Support publish/subscribe (one producer, many consumers) as well as request/response (one-to-one communication) interaction pattern among CAgents.
- Allow uniform serialization of messages exchanged among CAgents.
- Support communication bindings for different programing languages (C++, python, Java), so CAgent implementations can be language-independent.

One initial option that was considered is to use "raw" TCP sockets for inter-CAgent communication as in a preliminary UCB reference implementation introduced in [2]. However, several features would have to be added, such as many-to-one communication or message queuing functionality to allow adequate communication. Instead of using raw TCP connection and add custom features, we have selected the *ZeroMQ*[3] socket library to provide the CAgents with access to the VCC. ZeroMQ carries messages across TCP/IP, it allows one-to-many as well as one-to-one connectivity and realizes pub/sub as well as request-reply interaction patterns. It currently offers binding for 30+ languages including C, C++, Java, .NET, Python. ZeroMQ is open-source free software and runs on most operating systems, including Linux, Windows and OS X. It is very scalable and realizes fast message transportation. These features in combination with the natural decoupling of pub/sub clients and internal message queuing capabilities make the ZeroMQ socket interface a very suitable abstraction of the VCC.

While ZeroMQ realizes a socket library to access the VCC, it does not define serialization of messages (the process of converting a data structure or object state into a format that can be communicated among different CAgents). However, serialization is a requirement to correctly interpret the (e.g. CREW spectrum sensing) information on the receiver-side CAgent. We suggest using Google *Protocol Buffer*s[4] to serialize data structures exchanged among CAgents. Protocol Buffers are efficient because, in contrast to markup languages like XML, they allow to exchange data structures in a compact binary format. Since CAgents will likely exchange a large number of spectrum sensing information, efficient data exchange is an important precondition. However, Protocol Buffers are not self-describing, instead a specification of the message content (schema) has to be known by the recipient. As future work we plan to extend the CB framework with a device/service discovery mechanism, which can be used to query the schema (a Proto Definition file (.proto)); until then Proto Definitions have to be known statically by the involved CAgents. Like the ZeroMQ library Protocol Buffers are available for a large number of programming languages, such as C++, Java or python.

[3] http://www.ZeroMQ.org/

[4] http://code.google.com/apis/protocolbuffers/

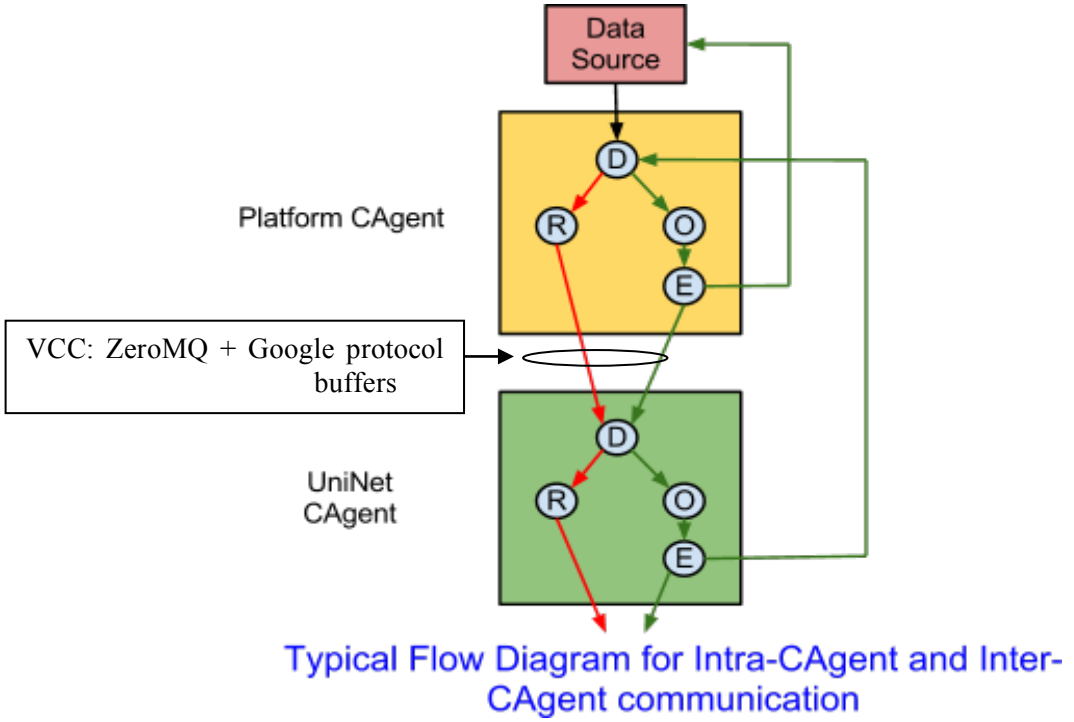

Figure 3: Inter-CAgent communication via VCC

Summary: CAgents access the VCC via *ZeroMQ* pub/sub and request/reply primitives and the exchanged data structures are serialized via Google *Protocol Buffer*s, which currently have to be known by the involved CAgents.

2.1.2.2 Intra-CAgent Communication

Intra-CAgent communication refers to the exchange of information among a CAgent's functional blocks (discovery, optimizer, executive, etc.). In general, we believe that these interfaces should remain *implementation-dependent*, as the components will be implemented on very different clients with varying programming languages and concepts; furthermore, for efficiency-reasons, blocks that are conceptually separate might even be realized in one single software components (e.g. a combined optimizer and executive object). Still, we formulate a set of guidelines on how to realize the intra-CAgent information exchange.

The easiest way to exchange data among CAgents functional blocks is via well-defined procedures (methods/functions calls). This may be the preferred choice if components are implemented in the same programming language (or language bindings are available). An example may be a *write*()-method provided by the Repository object and used by the Discovery object to store spectrum sensing information in the repository.

However, this mechanism requires that caller and callee have matching requirements relating to the kind of call execution (asynchronous / synchronous). For example a "write"-method on the Repository object may be blocking (synchronous), but the caller process (e.g. Discovery object) may require an asynchronous (non-blocking) write operation. In this case, method-calls are not suitable; instead, CAgents functional blocks should be implemented as independent stream of instructions that can be scheduled to run as such by the operating system (i.e. functional blocks are implemented as separate threads or different processes). In such a case we propose to use ZeroMQ also for intra-CAgent communication, because ZeroMQ enables efficient interprocess communication (IPC) with implicit queuing mechanism.

Summary: intra-CAgent information exchange is implementation-specific. If possible, objects communicate via well-defined procedures (methods/functions calls); whenever decoupled (concurrent) inter-process communication is required, *ZeroMQ* library will provide efficient IPC communication.

2.1.3 Implementation

In this section we describe our implementation of the CB framework structured by the internal CREW usage scenarios.

2.1.3.1 CREW usage Scenario 1 (Context awareness)

In this usage scenario we focus on gathering information on the surrounding spectrum usage and apply this knowledge to optimize the spectrum usage. To obtain the information on the spectrum usage we use, amongst others, the IMEC sensing engine, e.g. to measure the power in the DVB-T channels to tune the spectrum occupancy in the TVWS predicted based on a channel model as described in D6.2 or to improve the coexistence in the ISM band. In Figure 4 we show how the CB terminology applies to the ISM band scenario.

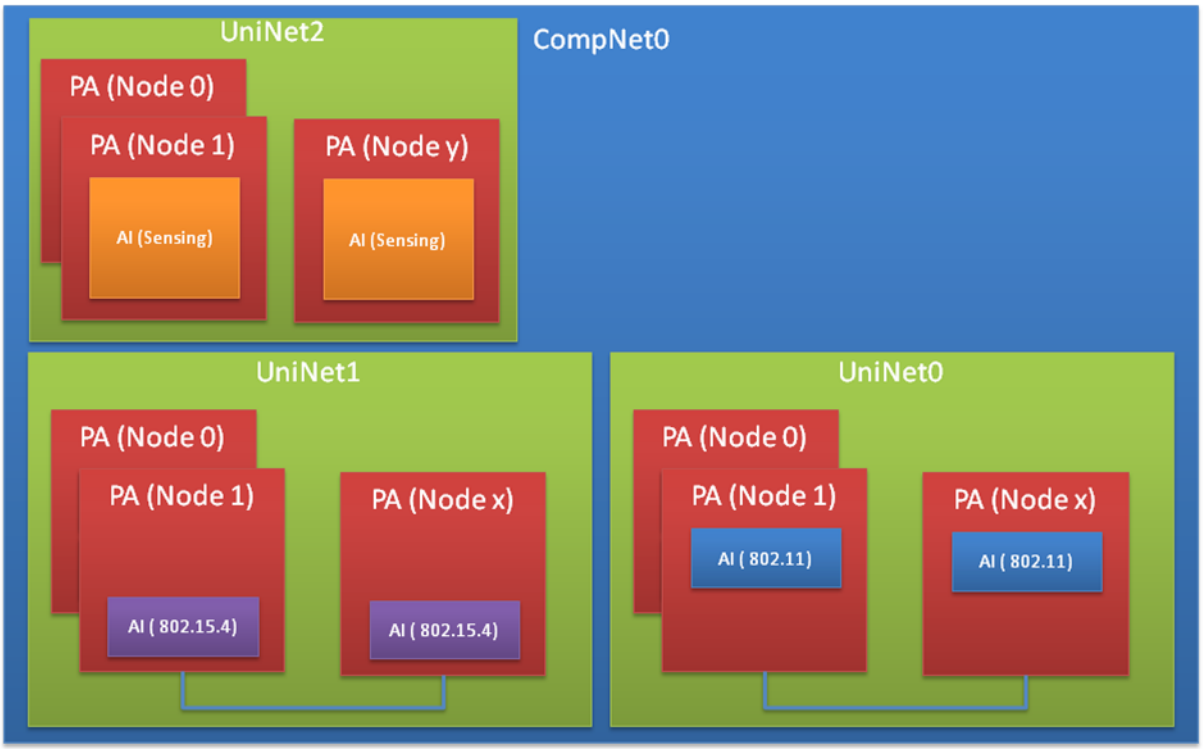

Figure 4: CB view on context awareness in the ISM band

For the CompNet to be able to optimize the overall performance it needs to be able to control the different UniNets, which requires a communication link. In a first phase we will focus on the communication link to and from the sensing engine. Within CREW we have selected the ZeroMQ layer and Google protocol buffers to implement the communication between different components. For the sensing engine this requires a layer on top of the existing API that handles the communication with the ZeroMQ layer and the translation of the Google protocol buffers. The communication with the sensing engine is bi-directional, configuration and control commands needs to go to the sensing engine and the measurement results need to go to the component controlling the sensing engine. For the protocol buffer containing the configuration command we use parameters in the sensing engine

API[5], for the protocol buffer containing the measurement result we foresee a data type with maximum 35328 floating point values, which covers all existing results that can be obtained from the sensing engine.

2.1.3.2 CREW Usage Scenario 2 (Robust Cognitive Networks)

The main objective of this usage scenario is the investigation of the robustness of cognitive radio solutions in cognitive Sensor Networks (CNSs) in order to achieve a certain QoS, while ensuring non-interference. Our particular focus is on body area networks (BAN) and building automation sensor networks. The CB framework provides a means to connect the different networks and components to exchange information, for example related to spectrum utilization.

Consider a Body Area Network (BAN) entering a building and moving through it. Assume the building infrastructure to have various wireless devices installed (these may be different technologies). In order for the BAN to coexist with these devices and maintain minimum mutual interference and maximum spectral utilization it is necessary for the BAN to be frequency agile or in other words it should possess cognitive radio abilities. A classic approach would mean the BAN does spectrum sensing by itself and decides what channel is suitable for it to transmit (*local spectrum sensing*). For a body area network, which has strict energy constraints, this approach is very costly to adopt. Alternatively, sharing spectrum sensing with an existing building automation network can make spectrum sensing more efficient and avoid energy due to spectrum sensing in the BAN. We call the interaction between a building automation network and a BAN with the goal to improve spectrum sensing in the BAN *infrastructure-supported spectrum sensing*.

In order to realize *infrastructure-supported spectrum sensing* in a BAN a framework is required that allows us handle inter-network communication between BAN and building automation sensor network to exchange sensing information. The framework should be such that it can be implemented in multiple use cases and is not implementation specific. To this end we adopted the CB framework in usage scenario 2 (US2) to have the means to experimentally investigate how in improving cooperation, collaboration and adaptivity of various wireless devices can improve communication robustness.

To this end we have implemented the CB framework on the TWIST sensor network testbed as well as on body area networks. The following Figure 5 gives an overview of the involved components:

[5] http://www.crew-project.eu/sites/default/files/SensingEngine_UserManual.pdf

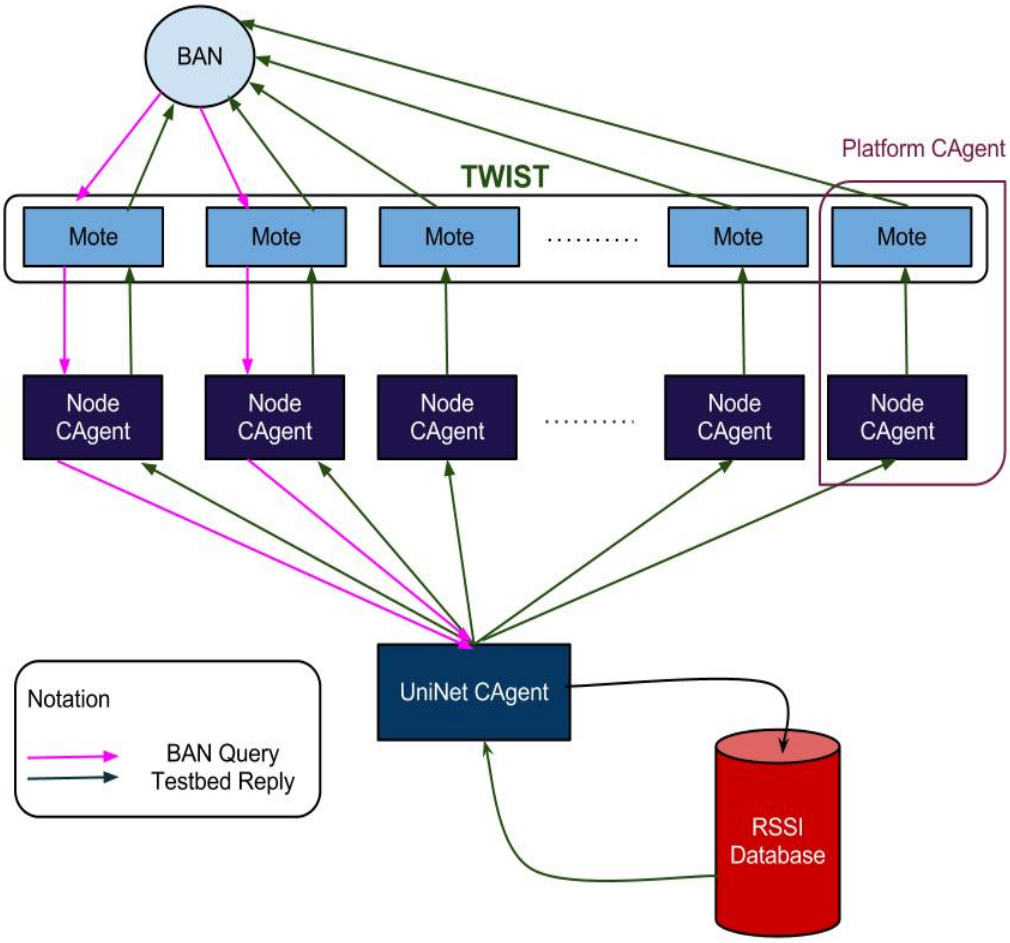

Figure 5: Mapping of US2 Scenario to the CB Architecture

The sensor nodes in the wireless sensor testbed, TWIST, perform spectrum sensing. Each TelosB [3] TWIST node is realized as a platform CAgent and has a single IEEE 802.15.4 Air interface CAgent associated with it. Both CAgents are implemented (to large extent, see below) via TinyOS[6] [4] components running on the TelosB nodes. However, as these CAgents have very limited storage resources, they cannot store the spectrum sensing results. Instead we introduced another layer of CAgent (a UniNet CAgent) that represents the collection of TWIST sensing nodes. All individual TelosB nodes publish their spectrum sensing results to the repository of the UniNet CAgent, which resides on the CREW server, where all data is stored. However, the TWIST sensor nodes are connected over serial interface (USB) to the TWIST infrastructure. Note that the serial protocol is TinyOS specific, therefore parts of each sensor node CAgent need to be implemented on a (CREW) server to connect to the nodes via the TinyOS-specific serial protocol handler (*SerialForwarder*) and unmarshal the data. Thus each sensor node CAgent consists of TinyOS components that realize the main CB functionality (sensing) as well as an object, running on the CREW server, that connects to exactly one node and provides a ZeroMQ (over TPC/IP) connection to other CB components.

The typical use case for CREW usage scenario 2 is as follows: the TWIST nodes continuously send spectrum sensing information to the NodeCAgents via the *SerialForwarder*. Each of the NodeCAgent sends the sensing information to the UniNet CAgent via the VCC. The UniNet CAgent listens on the VCC and stores the information learned in a global database handled by the repository of the UniNet CAgent. When an external BAN enters the network space, it queries to the testbed, expecting a reply that contains information about which channels in the 2.4GHz band are suitable for it to transmit. The

[6] http://tinyos.net

NodeCAgents forwards the query to the UniNet CAgent, which retrieves information from the database, executes the CB optimization algorithm and broadcasts optimized result back to the BAN via the NodeCAgents and the testbed motes.

The CB framework can be expressed well with the object oriented programming paradigm. We implemented the CB components for CREW US2 in the C++ programming language. Note, however, that since the VCC (interconnection among CAgents) is language-independent our implementation can interact with CAgents developed with other programming languages.

In the following we first present the UML diagram of our design, shown in Figure 6. This diagram focuses mainly on the CAgent base class and its components. Later in Figure 7, we present our class diagram involving the NodeCAgent (part of our Platform CAgent), which we call TWISTNodeCAgent and UniNetCAgent, which we call TWISTTestbedCAgent. Note that the RSSI database that we mentioned in our usage scenario is implemented using MySQL in our implementation and we use MySQL C++ connector libraries in-order to handle the database.

UML Diagram

The CAgent class in our design has 4 members Repository, Optimizer, Discovery and Executive (Figure 6). `VirtualControlChannel` is the basic class that defines ZeroMQ sockets, sets up the PUB/SUB links of the VCC and provides methods for serializing data (from integer array to protobuf and vice versa). This may be the base class or a member variable of any CAgent component that needs VCC support according to the CB framework. For a component needing only single VCC socket, it can be derived from VCC class (For example - `Repository` and `Executive` class) and for a component needing multiple VCC sockets, it is convenient to make VCC objects as member variables (for example `Discovery` class).

Note that there are a number of variables/methods defined for each CAgent block, but not all are needed for a CAgent of particular type. It depends on how the CAgent blocks are initialized to decide which functions they utilize. In the following we briefly discuss the methods provided by each of the CAgent blocks.

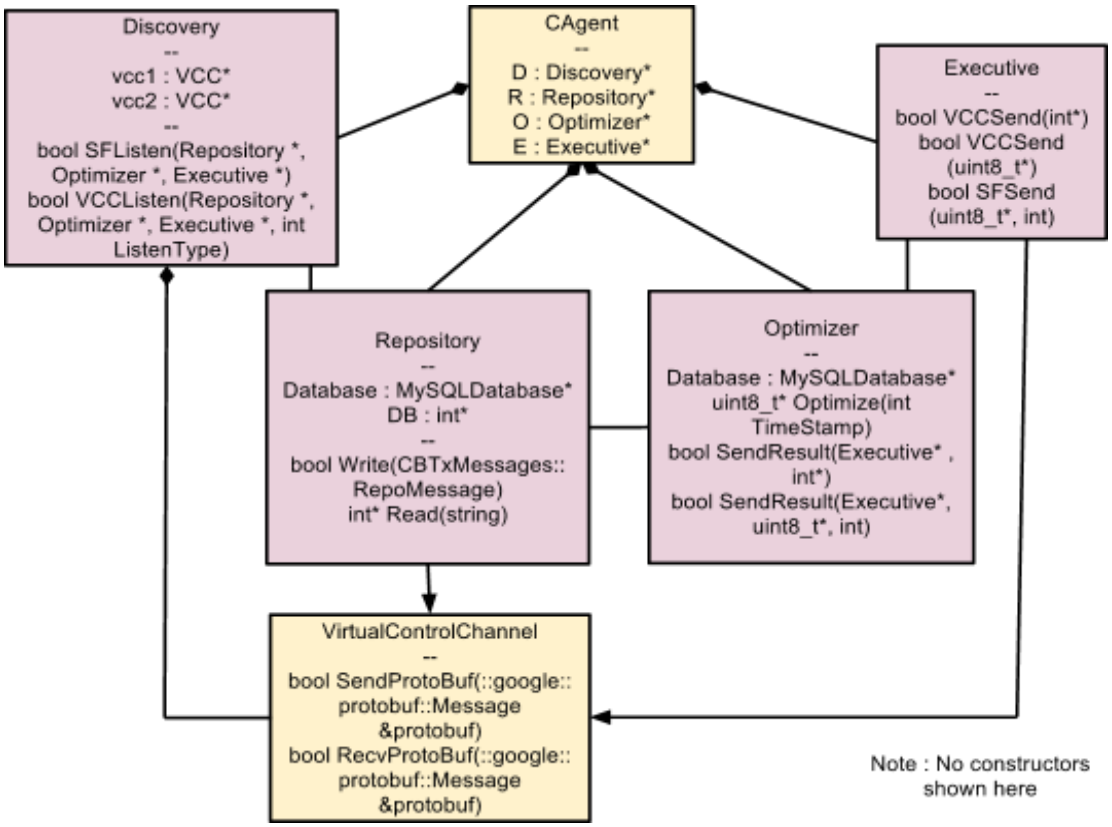

Figure 6: UML Diagram of the CB Implementation for US2

`Discovery`

This component provides methods for listening for the sensing information over the serial forwarder (connected to a sensor node) as well as the VCC.

`Repository`

This component contains the database as a member variable. For `TWISTNodeCAgent`, we define a single entry integer array as a dummy database since the sensor nodes do not store sensing data locally. For `TWISTTestbedCAgent`, we define a MySQL Database. Repository provides two basic methods - `Read()` and `Write()`. These functions have been made as generic as possible. `Read()` function takes one argument which can be any `SELECT` command in MySQL for reading data from a table. The `Write()` function on the other hand, first updates the database and then publishes the updated information on some VCC endpoint.

We have defined a `MySQLDatabase` class (not shown in the diagram), that defines insert, update, delete and retrieve functions for handling MySQL databases using C++. This class is used for achieving the `Read()` and `Write()` functions. If it is decided to utilize the Repository class using a different database handling mechanism, it can be easily done by defining a new Database class similar to MySQLDatabase class and providing insert, update, delete and retrieve functions to this class.

The MySQL database, which the `Repository` handles, basically has two tables. The first one stores information about the testbed nodes like - node ID, location coordinates, type of platform etc. The second one stores the RSSI value corresponding to a node ID along with a unix timestamp. This table not only stores the current RSSI values but also stores a record of previous entries. The number of backup entries can be controlled by restricting the maximum size of the table that can be afforded.

`Optimizer`

The `Optimizer` has access to the database, which is updated by the Repository. It provides `Optimize()` method that calls an optimization algorithm and stores the result in an uint8_t array. Note that this algorithm is very much implementation specific. In our implementation, there is no algorithm required in the `TWISTNodeCAgent` Optimizer. We implemented a simple 'global averaging' algorithm for the `TWISTTestbedCAgent` Optimizer.

The 'global averaging' algorithm is implemented as follows - the user specifies over what time interval does he want to optimize and then the RSSI values for all channels that fall in that time range are selected and averaged over all nodes, generating a 2 byte result where each bit corresponds to a channel in the 2.4GHz ISM band. Bit value 0 corresponds to a congested channel indicating that the external BAN should not transmit in that channel. It is clear that the algorithm we implement is not accurate for our usage scenario (as position of the BAN should be taken into account while finding the optimized result) but is implemented just to demonstrate the usage of the CB framework developed. Formulating a localized algorithm for optimization surely gives a scope for future work. Naturally, CREW experimenters can implement more elaborated versions `Optimizer` algorithms.

`Executive`

This component provides function for sending different kinds of messages (repository message/ optimizer query/ optimizer reply) over VCC. It also provides function `SFSend()` to send messages to the sensor nodes via the TinyOS SerialForwarder.

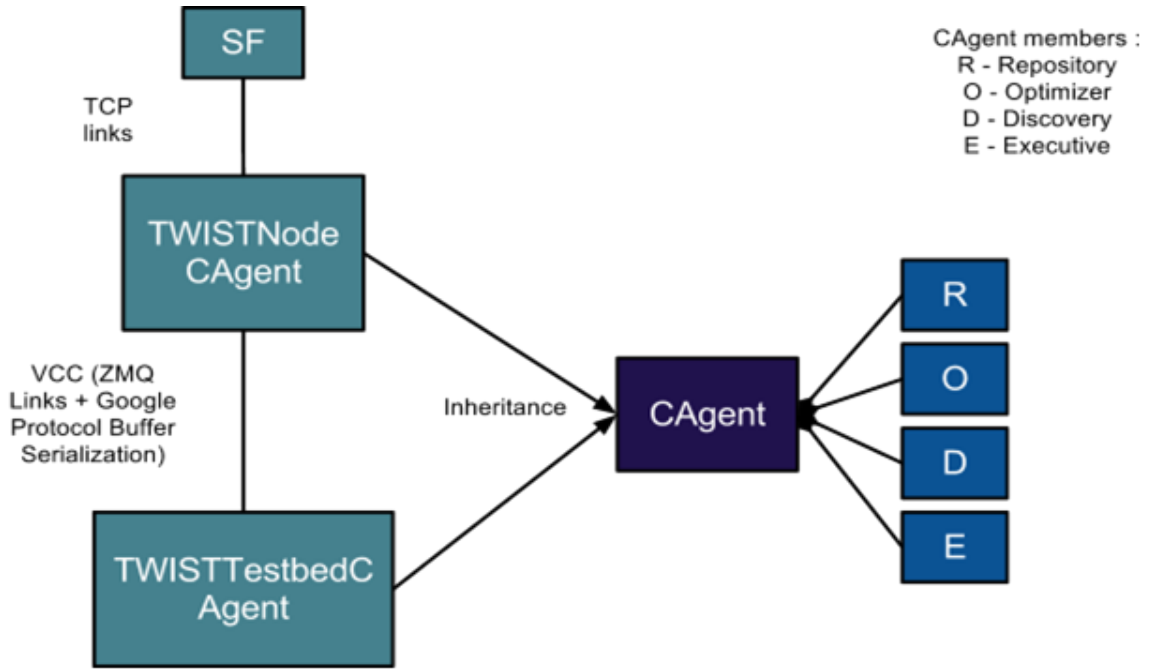

Figure 7: Class Diagram

Our usage scenario demands multiple `TWISTNodeCAgents` to send data to a single `TestbedCAgent`. Similarly we also want all the `TWISTNodeCAgents` listen to the optimized message from the same `TWISTTestbedCAgent`. The following Figure 8 explains how to implement such fan-in and fan-out using ZeroMQ PUB-SUB sockets.

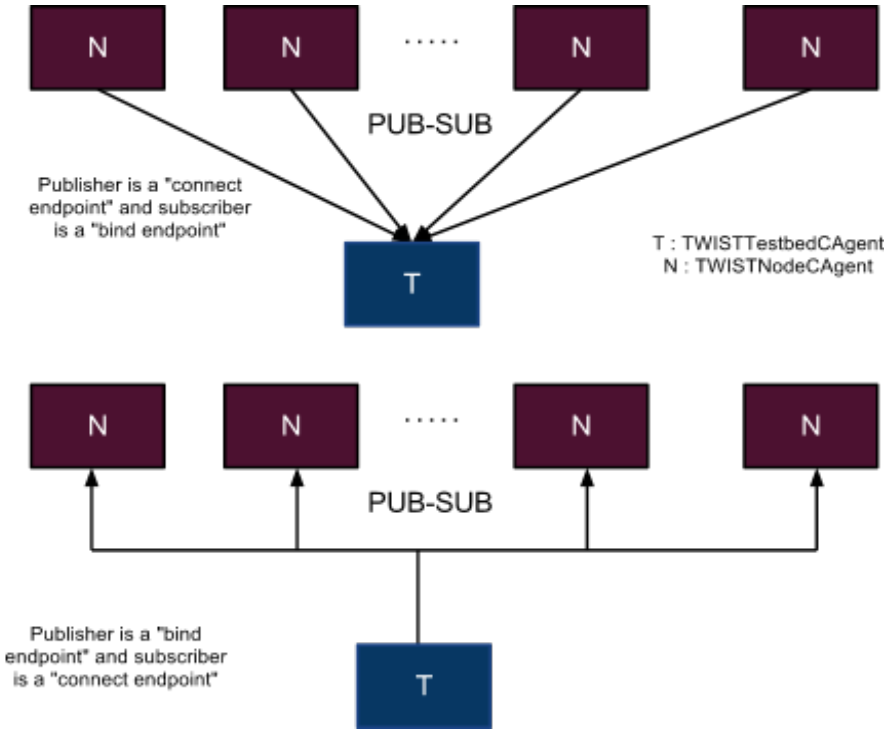

Figure 8: Implementing fan-in and fan-out using ZeroMQ, PUB/SUB sockets

From this we can derive that we need the following VCC support for the CAgent blocks:

`Repository` : ZMQ_PUB, TCP - (1) connect endpoint

`Executive` : ZMQ_PUB, TCP - (1) bind endpoint

`Discovery` : ZMQ_SUB, TCP - (1) connect endpoint and (2) bind endpoint

`Optimizer` : No need of ZeroMQ support

If IPC is used for Intra-CAgent communication the `Optimizer` would need ZMQ_SUB, IPC and ZMQ_PUB, IPC support; `Repository` would need ZMQ_SUB, IPC support; `Executive` would need ZMQ_SUB, IPC support and `Discovery` would need ZMQ_PUB, IPC support. However, intra-CAgent in our implementation is realized simply via method-calls.

In the following we list two sets of examples where we explain what parts of our CAgent implementation react how to which events and which are the interfaces provided.

Example 1: Motes send spectrum sensing information to NodeCAgents

Event	**Who is in charge?**	**Function**
Listen to serial forwarder	Discovery of	SFListen()

	NodeCAgent	
Identify message type as Repository Message	Discovery of NodeCAgent	-
Post learned information to Repository	Discovery of NodeCAgent	IPCSend() / simple method call
Repository stores information in its own Database	Repository of NodeCAgent	UpdateDatabase() (Implementation Specific)
Repository publishes some part of its database over VCC	Repository of NodeCAgent	VCCSend()
Listen to incoming messages over VCC	Discovery of TestbedCAgent	VCCListen()
Post learned information to Repository	Discovery of TestbedCAgent	IPCSend() / simple method call
Repository stores information in its own Database	Repository of TestbedCAgent	UpdateDatabase() (Implementation Specific)
Repository publishes some part of its database over VCC	Repository of TestbedCAgent	VCCSend()

Example 2: Sensor nodes send BAN Query to NodeCAgents

Event	**Who is in charge?**	**Function**
Listen to serial forwarder	Discovery of NodeCAgent	SFListen()
Identify message type as Optimizer query	Discovery of NodeCAgent	-
Post learned information to Optimizer	Discovery of NodeCAgent	IPCSend() / simple method call
Execute Optimization algorithm (in our usage scenario this is NULL)	Optimizer of NodeCAgent	Optimize() (Implementation Specific)
Pass the optimized message to Executive	Optimizer of NodeCAgent	IPCSend() / simple method call

Send to interested party (in this case it is higher level CAgent)	Executive of NodeCAgent	VCCSend()
Listen to incoming messages over VCC	Discovery of TestbedCAgent	VCCListen()
Post learned info to optimizer	Discovery of TestbedCAgent	IPCSend() / simple method call using friend classes
Execute Optimization Algo	Optimizer of TestbedCAgent	Optimize() (Implementation Specific)
Pass the optimized message to Executive	Optimizer of NodeCAgent	IPCSend() / simple method call
Send to interested party (in this case it is lower level CAgent)	Executive of NodeCAgent	VCCSend()
Listen to incoming messages over VCC	Discovery of NodeCAgent	VCCListen()
Post learned info to optimizer	Discovery of NodeCAgent	IPCSend() / simple method call using friend classes
Send to interested party (in this case it is Testbed motes)	Executive of NodeCAgent	SFSend()

Our implementation can be reused and easily extended by CREW experimenters, but the following points must be considered while extending our CB based design for a different usage scenario:

- `VirtualControlChannel` class may require changes in `ConvertToProto`() and `ConvertFromProto`() functions, which depend on the .proto message classes defined. Apart from this subtle changes like changing PUB socket to a bind endpoint or a connect endpoint may be required.
- MySQLDatabase is an implementation specific class. The developer might need to modify its methods for the repository name, table names, table column names and their size. If some other kind of database is to be managed, it is necessary for the developer to design a similar class for handling the new database by providing it the necessary retrieve, insert, update and delete methods.
- The number of VCC sockets required for `Repository`, `Executive` and `Discovery` may change as per the new implementation requirement. Thus, the constructor initialization of these blocks will change accordingly.
- If the message format is changed, there may be a need to synchronize the new message format (may be repository message or optimizer message) with the .proto files and also subtle changes in the `UpdateDatabase`() function may be required.
- The `Optimizer` algorithm is very much implementation specific and changes with usage scenario.

- The new scenario may demand the CAgent components to run in parallel on different threads. In this case, it may be required to introduce IPC (inter-process communication) using ZeroMQ sockets. In this case the VCC class can be directly used for the same, as just the endpoints now change from TCP to IPC and the `SendProtoBuf()` and `RecvProtoBuf()` functions remain the same. Note that in this case you may consider changing the name of VCC class to `ZMQLinks` or any other suitable name, as IPC is not theoretically a part of VCC.

2.1.3.3 CREW Usage Scenario 3 (Horizontal resource sharing in ISM bands)

The CREW US3 involves a predefined home environment, including several WiFi and Bluetooth devices. Besides the devices for generating traffic, some sensing devices are also deployed to monitor the environment, such as IMEC sensing agent, WiSpy or USRPs. Finally, a sensor network is employed to emulate the home automation scenario. One example topology of emulated home scenario is shown in **Error! Reference source not found.**, where red dot represents the sensor network. Different scenarios can be defined. In a first scenario the normal sensor network performance is measured. In a second scenario the sensor nodes are assisted by a distributed heterogeneous sensing platform for selecting an optimal receive channel and the network performance of the sensor network is measured again..

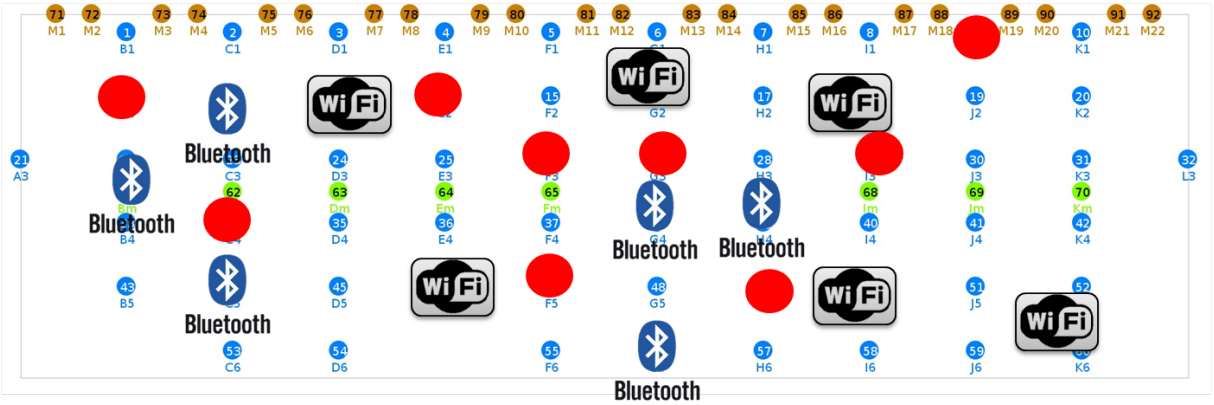

Figure 9: Devices in CREW US 3

The collection of experiment results, the set-up and iteration of measurements and the communication between cognitive sensor network and distributed sensing network are achieved via the CREW benchmarking frame work.

Scenario 1: Normal sensor network

The component mapping towards the connectivity brokerage for the normal sensor network is presented in the Figure 10:

- CompNet0 is the w.iLab.t testbed (pseudo-shielded environment)
- UniNet0 represents the sensor network under test. The sensor nodes use Zigbee as the air interface

- UniNet1 represents the spectrum sensing engine networks, it may be made up of several sub UniNets. One sub UniNet1.1 is the IMEC spectrum sensing network. Another sensing network UniNet1.2 could be USRP spectrum sensing network or WiSpy. Each sub UniNet is a distributed spectrum sensing network, focusing on different frequency ranges or different technologies.

By linking together the sub UniNets, a distributed and heterogeneous spectrum sensing engine is formed. In this first experiment, the spectrum sensing network is only responsible for monitoring the wireless environment. The sub UniNets 1.1 and 1.2 mainly perform the "Repository" and "Discovery" functionality. UniNet2 and UniNet3 represent a WLAN network and a Bluetooth network respectively. The Platform Agents for WLAN network are typically wireless routers and laptops, while the Platform Agents for Bluetooth network can consist of very diverse devices, e.g. Bluetooth head sets or smart phones. The AI for UniNet2 and UniNet3 are obviously WiFi respectively Bluetooth.

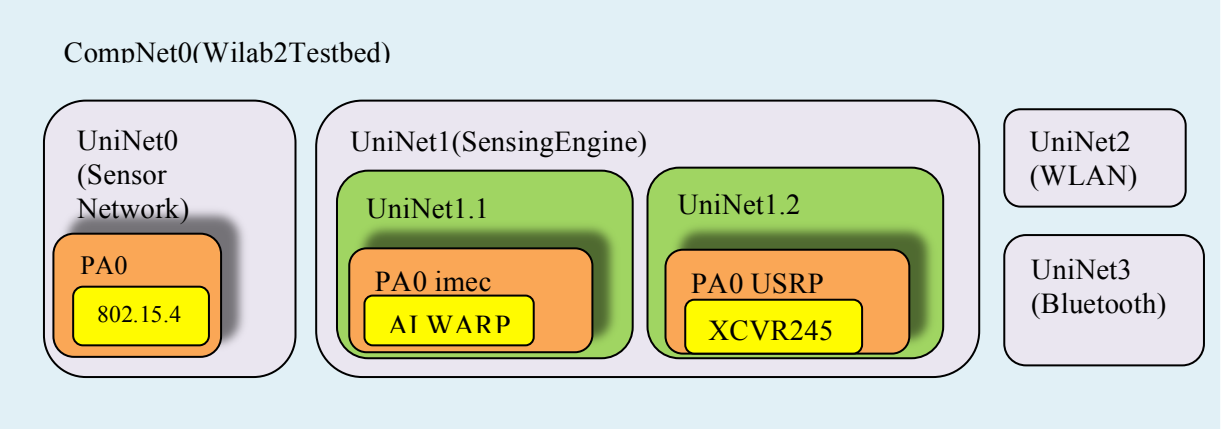

Figure 10: Mapping of US3 to CB Framework

Scenario 2a: Cognitive sensor network (local sensing)

The easiest way to form a cognitive sensor network is to perform channel evaluation based on local noise measurements by the sensor nodes themselves. Thus the component mapping of CB stays basically the same as the none-cognitive network, except that UniNet0 now also implements spectrum sensing and decision making functionalities.

Scenario 2b: Cognitive sensor network (distributed sensing)

In a more advanced cognitive sensor network setting (see Figure 11), UniNet0 can cooperate with UniNet1 to get spectrum sensing information from the wireless environment. Thus, an extra CompNet1 is formed. The functionalities of CompNet1 are administration and communication between UniNet0 and UniNet1.

In summary, Uninet1.1 and 1.2 perform spectrum sensing measurements with different channels and gain settings ("Discovery" functionality in the CB concept) and each have their own repository ("Repository" functionality). Uninet1 combines the spectrum information for its sub Uninets and further translates the spectrum information into a decision on the optimal sensor network settings (receive channel settings), equivalent to "Optimizer" functionality. CompNet1 disseminates the information from UniNet1 to UniNet0 and is hence responsible for the "Executive" functionality in CB concept.

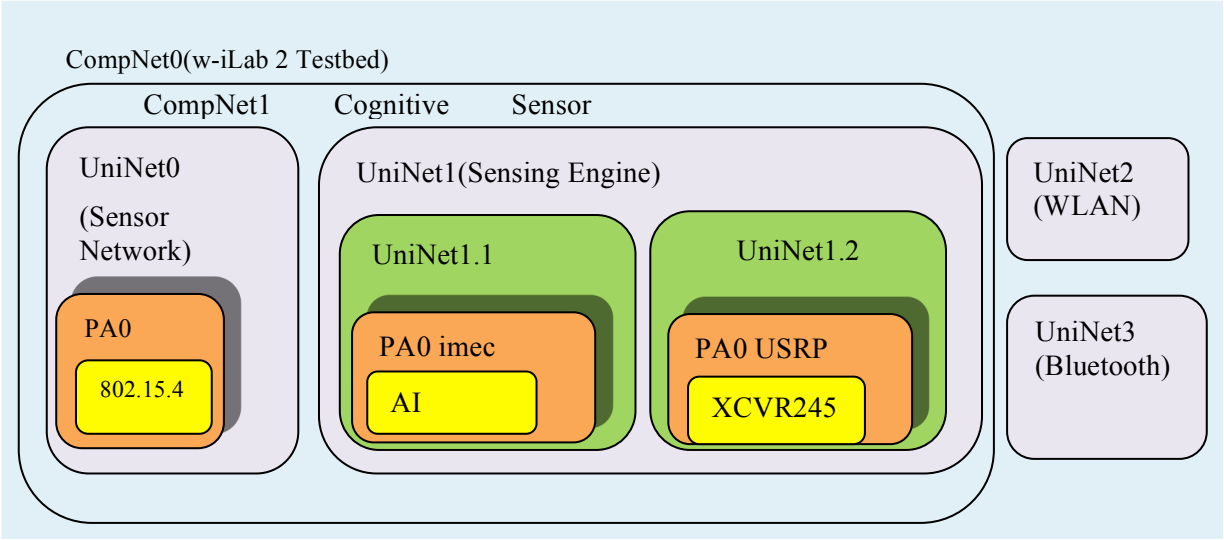

Figure 11: Mapping of US3 to CB Framework (Distribited sensing)

Due to time limitation we mainly implemented the scenario 2b, where channel allocation is based on distributed spectrum sensing. Further information is provided in D6.2 section 2.3.1.

JSI Testbed

The VESNA based LOG-a-TEC testbed can support Connectivity Brokerage in a similar manner as the TWIST testbed described in Sect. 2.1.3.2. The implementations of the Air Interfaces, Platform Agents, UniNets and CompNet can run on the infrastructure serving the testbed and communicate with the hardware by using the testbed-specific interface and communication protocol.

For experiments, the same interface can be provided as for the other testbeds in CREW. However, the implementation of CAgents on VESNA nodes needs a different approach, because the ZeroMQ protocol, used as transport layer in CREW for inter-CAgent communication, is not supported on microcontrollers. In case there is a demand for CAgents to run on the devices deployed in the LOG-a-TEC testbed, various communications protocols can be considered as alternative, for example CoAP, and the used protocol can then be translated to the standard inter-CAgent protocol transparently.

2.1.4 Conclusions and Future Work

We have adopted the *Connectivity Brokerage* (CB) [1] architecture in CREW in order to allow CREW experimenters to integrate their mechanisms-under-test in a well-specified CR framework and facilitate experimentation by enabling reuse of CR software components and concepts. To this end we first analysed and identified the limitations in the CB framework; we then suggested CREW-specific solutions, namely the use of ZeroMQ library and Google protocol buffers to enable interconnection of CB objects (VCC) on different hosts and written in different programming languages. We then described our implementation, which covers different CREW usage scenarios. During this process we discovered topics for future work:

- Some CREW scenarios involve mobility, it is not clear if the current framework provides enough support for this (e.g. how the VCC can be maintained when crossing networks).

- The CB framework lacks detailed specification of how to react in case e.g. messages are received over an AirInterface, and their semantic might by unknown by the receiving CAgent (e.g. such message could be forwarded to a default gateway/higher level CAgent).
- It is not clear if the Publish/Subscribe interaction pattern over the VCC is the optimal solution; in fact for CREW US2 we identified cases where direct one-to-one communication seems more efficient. A more systematic investigation is required and a set of guidelines for selection of interaction patterns seems useful. Furthermore currently TCP/IP communication is necessary to take part in the CB framework; it has to be analyzed if other solutions may be used (especially when resource-constrained devices like sensor nodes are used, UDP/6LoWPAN may be more suitable).

2.2 CREW-GENI collaboration: Joint development of a common cognitive radio language

During Year 2 of the CREW project, a collaboration has been established between CREW and GENI through various bilateral discussions between CREW and Ivan Seskar from WINLAB /Rutgers University. WINLAB is one of the leading partners in the GENI COGRADIO[7] project. The collaboration between CREW and GENI has now been formalized (see support letter in Appendix 2), the actual collaboration will start in October 2012 (Year 3 of the CREW project).

The main motivation for the CREW-GENI collaboration is given by the following observations. Due to the continuous growth in the number and density of wireless devices, standard wireless solutions suffer from spectrum bottlenecks and coexistence problems. In order to use the scarce spectral resources more efficiently, there is a need for more flexible and more intelligent radios, also called cognitive radios (CR). Today several hardware and software platforms for CR research are available on the market. Some of these platforms are deployed in experimental CR test facilities like those in the CREW project or the in the European FP7 GENI COGRADIO project in the US. A few examples of popular CR platforms deployed in experimental facilities are:

- USRP hardware[8] in combination with GNU Radio software[9] or the IRIS modular software platform developed by TCD[10]
- WARP1[11]
- GENI CRKit[12]
- IMEC spectrum sensing agent[13]

Although experimental facilities are open and readily available, the learning threshold for setting up end executing a CR experiment is often quite high. Each of these complex CR platforms requires investing a lot of time and effort to get familiar with detailed specifications in order to conduct even

[7] http://groups.geni.net/geni/wiki/COGRADIO

[8] Ettus Research, https://www.ettus.com/

[9] http://gnuradio.org/redmine/projects/gnuradio

[10] http://www.crew-project.eu/iris

[11] Rice University, http://warp.rice.edu/

[12] http://crkit.orbit-lab.org/

[13] http://www.crew-project.eu/portal/imecdoc

the most basic experiment especially on a larger scale (i.e. complex topologies consisting of more than two nodes).

2.2.1 Goal of the collaboration

The CREW-GENI collaboration targets the joint development of a common CR language to facilitate experimental validation of advanced CR solutions. Through the CR language, experimenters will no longer need to know all technical details and software interfaces of individual CR platforms they plan to use and as such they can better focus on their individual research goals.

One of the use cases investigated within CREW is distributed heterogeneous spectrum sensing, where we want to establish a dynamic map of power spectrum density (PSD) via distributed heterogeneous devices. At the same time, in ORBIT, a range of spectrum sensing solutions were developed to support opportunistic rendezvous types of CR applications. In the first phase, we therefore aim to design a common set of CR language statements to configure heterogeneous spectrum sensing devices in view of collection, processing and dissemination of spectral information. We see two interesting usage scenarios for distributed spectrum sensing:

(1) spectrum sensing as part of the solution to realize true cognitive networking;
(2) spectrum sensing as an experimentation tool to assess the environment during wireless experimentation.

A second phase, focused on expanding the CR language to support other types of configurations and experiments, may be defined later upon completion of the first phase. This would include utilization of domain knowledge through use of RF, spectrum sensing, cognitive radio as well as system and testbed ontologies to support semi-automatic and automatic experiment generation.

2.2.2 Approach

We propose to implement the common CR language implemented on top of Experiment Description Language (OEDL), which is part of OMF (cOntrol and Management Framework)[14]. OMF is nowadays considered as the de facto standard for control, measurement and management of wireless testbeds. This framework is already used for more than 7 years in the ORBIT testbed[15], is one of the GENI control frameworks [16] as well as primary management platform for the GENI COGRADIO project. In the CREW project (a part of[17]) the IBBT test facilities are already controlled with OMF. For other facilities, like the TWIST testbed @ TUB and the Iris testbed @TCD, OMF is in the roadmap.

We propose to build further on the Connectivity Brokerage (CB) concept [1]. The main goal of CB is to provide a general framework that enables diverse wireless technologies to exchange information and collaborate in a seamless fashion. We refer to the CREW internal document "Working_Document_WP5_Connectivity_Brokerage" for more information on the CB concept and on how the CB concept can be applied for building a common API for (re)configuring CR nodes.

The following functionalities needs to be supported

[14] http://mytestbed.net/

[15] ORBIT testbed consists of 400 node indoor facility with variety of radio platform including USRP, USRP2, WARP and CRKIT, 8 smaller specialized development testbeds with 2 to 8 nodes each and outdoor testbed with 20 nodes spread thought two campuses in NJ.

[16] GENI WiMAX meso-scale deployments for wide-area wireless experiments include 10 sites throughout the US with 17 WiMAX basestations using OMF as their control framework.

[17] The IBBT wireless facilities consist of 2 parts: a first part is deployed across three floors of the IBBT office building in Ghent, Belgium. A second part is located in Zwijnaarde, Belgium, approximately 5 km away from the office lab. The second location hosts, in addition to standard ISM technologies, also software defined radio platforms (USRP) and spectrum scanning engines developed by imec. Only the second location is OMF controlled.

- *Easy set up of the CR platform*: This involves developing the OMF management tools for flashing/programming of the CR hardware (FPGA, microcontroller) and/or installation of the CR software.
    - IBBT plans to provide the OMF tools for setting-up the IMEC sensing agent and the USRP/IRIS platform
    - IMEC will extend sensing engine API to support the Connectivity Brokerage implementation within CREW in respect to the formatting of the messaging generated by the sensing engine device
    - GENI CRKIT set of OMF tools will be harmonized with CB based API and implemented OEDL CR language extensions.
    - Support will be extended to other platforms including VITA 49[18] based peripherals.
- *Runtime support of distributed sensing through a common API.* To this end a common subset of relevant reconfigurable parameters for different CR platforms will be identified. This will allow the at runtime adaptation of reconfigurable parameters, hereby abstracting the heterogeneity of the underlying hardware and software.
- *Development/Adoption of common spectrum sensing archiving format.*

The same common CR language will be demonstrated on facilities offered by the GENI COGRADIO and the CREW project. Envisaged demonstrations are:

- Heterogeneous distributed spectrum sensing as an observation/measurement tool during wireless experimentation using IMEC spectrum sensing agent, USRP/IRIS and IEEE 802.15.4 radio at IBBT w-iLab.t. Through distributed sensing we plan to assess the wireless environment during experimentation and detect anomalies. In order to maximize spectrum occupancy information collected from PSD measurements, it should be possible to reconfigure sensing devices at runtime.
- On-demand spectrum monitoring facilities for ORBIT testbed. The plan is to offer both on-line and off-line spectrum sensing and monitoring service to testbed users. In addition to monitoring, the developed framework will be offered as an experimental platform for further development of spectrum sensing techniques and applications.
- Multi-site large scale spectrum sensing demonstration using advanced networking capabilities of GENI and FP7 testbeds as a proof-of-concept infrastructure necessary to support wide area dynamic spectrum assignment research.

[18] http://www.vita.com/home/VSO/VSO.html

2.3 Hardware-related extensions

2.3.1 Extension of the TUB testbed

During several CREW experiments we had identified that it would be valuable to monitor the very crowded 2.4GHz ISM band during CREW experiments. To this end we have added a set of commercial, low-cost USB spectrum analyzers WiSpy-2.4x to the TWIST testbed. These spectrum analyzers can be used (1) before an experiment to check RF interference conditions and decide if and where (what frequency range) to start the experiment; or (2) during an experiment to validate the experiment conditions (RF interference environment).

The WiSpy devices are connected via USB interface to the "supernodes" of the existing TWIST testbed infrastructure. The TWIST supernodes are network-attached storage devices that are already placed in every room of the office building to provide access to the TWIST sensor nodes (reprogramming, interaction during experiments over a serial channel). A schematic overview of the involved components can be seen in Figure 12.

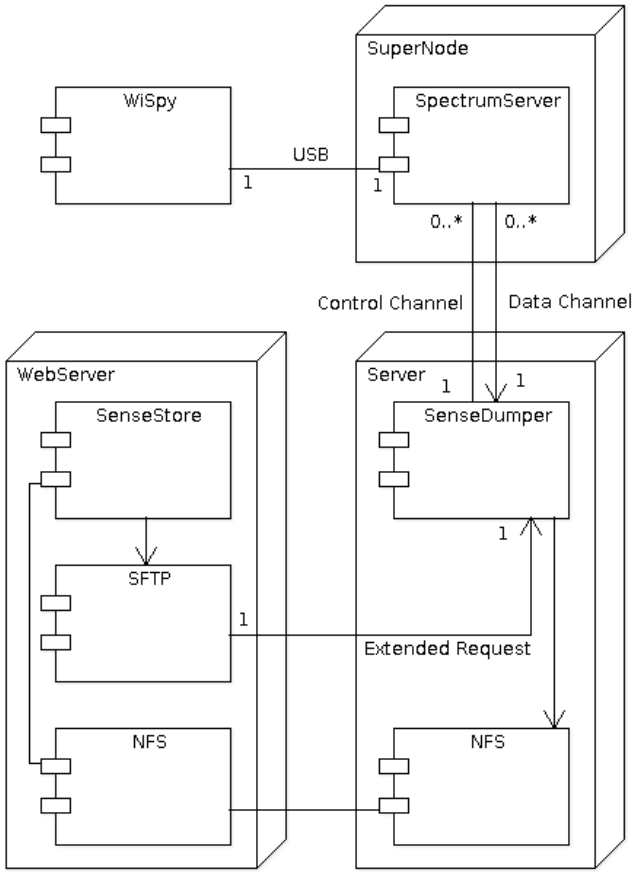

Figure 12: Components involved in the WiSpy integration into the TUB testbed

A WiSpy device sweeps the spectrum starting at 2.4 to 2.5 GHz by increasing the center frequency of a radio receiver by a given step size for a given number of times. The center frequency is held constant over a configurable time interval. Filtering of the receiver's output is done with an adjustable filter

bandwidth. This way the user obtains the signal power for individual frequency ranges with equal width. The WiSpy is capable of sweeping a range with some thousand steps with a minimal step size of 23.5 kHz and the filter bandwidth needs to have a minimal value of 53.6 kHz. The dwell time ranges from 10 µs to 2.55 ms in increments of 10 µs.

The parameters provided during configuration are always accepted however the used values might differ slightly. Especially the number of samples can be a bit higher or lower than specified. All effective parameter values are stored together with the sweep data. As only raw samples *S* will be stored the power values P need to be calculated using the following simple equation:

$$P\ [dBm] = S \cdot 0.5 - 134.0\ [dBm]$$

In TWIST a supernode is used to manage the operation of a set of (typically 4) sensor nodes and relay their serial communication channels. A software component developed by TUB, called "spectrumserver" enables set and read the parameters of a connected WiSpy and to pass the retrieved spectrum sweeps on to the TWIST server as well. Spectrumserver consists of several parts written in C++ of which the device driver, the control channel and the data channel are the most important ones. Each of them is polled within a control loop and is continuously checked for errors.

Initially the spectrumserver waits for a connection to the control channel over which commands are received. There are commands to set the parameters of the WiSpy, to start or stop sweeping, to disconnect and to retrieve actual parameters read from the device. The control channel protocol is text based and replies with a status code and a message explaining what is happening. A proper command session begins with setting the parameters, optionally also reading them afterwards and the start command which includes a port number for the data channel. A stop and disconnect command may end the session at any time.

The device driver encapsulates the management of the WiSpy and transforms any error into an exception. As the WiSpy communicates over USB as a Human Interface Device (HID) the open source library libusb is used to send HID compliant messages. USB 2.0 transfers all messages in frames with 64 bytes of payload so all sweep messages are received fragmented and passed as a complete message, which then is fed into the data channel. When the spectrumserver receives the start command it listens to the given port and once the connection is established it starts reading sweeps from the WiSpy, which are send over the data channel. As the data channel is more data intensive than the control channel it operates with a binary protocol, which sends a packet including a time stamp for each sweep.

Previously the TraceServer on the TWIST testbed only handled dumping output from the sensor nodes to a file. It has been extended to also control the spectrumservers and dump sweeps from all enabled supernodes into one file per job. The implementation of the TraceServer involved introducing a new dump manager, which takes care of job instances, which in turn handles the control routines for each spectrumserver. All data is dumped into a NFS replicated directory to provide the web server with the dump files. Finally, a new sensestore was introduced on the web server in analogy to the existing tracestore. It is responsible for managing all sweep dump files and issuing the control commands for the whole sensing infrastructure for an extension to the SFTP protocol which connects the web server to the TWIST server. Naturally the web interface had to be changed too to incorporate the controls for spectrum sensing and dump file retrieval.

To summarize: a distributed spectrum sensing system utilizing low cost spectrum analyzer hardware was successfully integrated into the TWIST testbed. This enables users of the testbed to check RF interference environment and validate their experiment conditions.

2.3.2 Extension of the IRIS testbed

For the Iris testbed the following demand driven extensions were identified and set out in task 5.1:

1. Expansion of the Iris testbed to incorporate additional nodes
2. Incorporation of GPS and GUI functionality

3. General support / extension of the Iris functionality

The first two of these are discussed here while the third is discussed within Section 3 as part of the discussion of support for external experimentation and open calls. It must be noted that due to the late start in the demand driven extension work due to delays in funding, parts of this work are still ongoing.

2.3.2.1 Expansion of the Iris testbed to incorporate additional nodes

The Iris testbed was extended to incorporate 10 additional nodes (currently there are 4) and also to incorporate MAC layer and GUI functionality as shown in the following Figure 13.

.

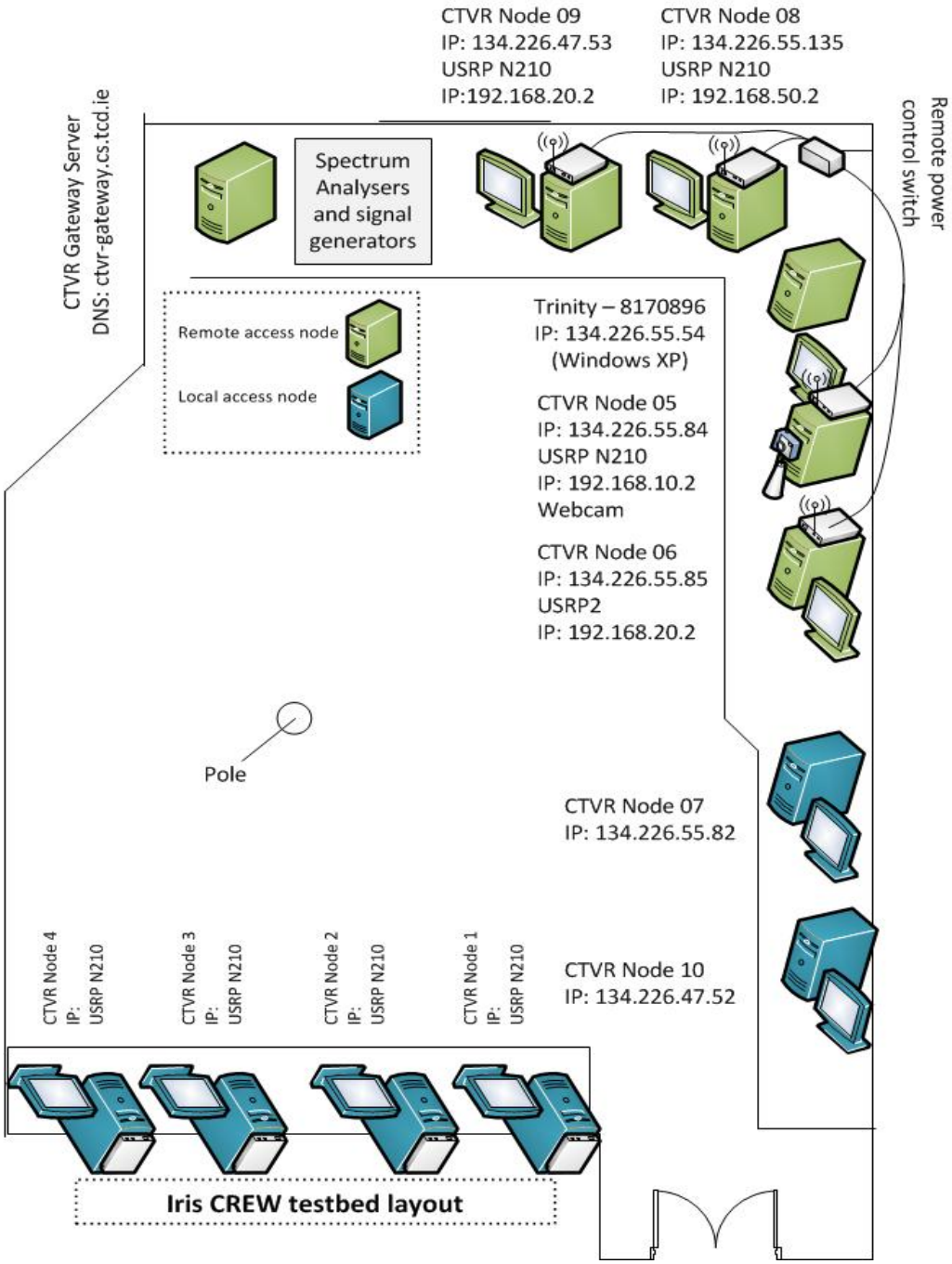

Figure 13: Extended Iris CREW Testbed

In order to accommodate the growing interest in IRIS and the testbed, four new nodes were purchased. This included four DELL T1500 desktops with Intel Xeon 3.4GHz Quad core processors (to allow for high complexity radio designs) and four USRP N210s. Each of these nodes is connected to the gateway node in the same fashion as the others to allow remote access. This addition brings the total nodes available for use by external users up to 8. As with the others, the use of the nodes is managed by a calendar booking system.

Furthermore all machines in the testbed have been updated and are now running Ubuntu 12.04. Each also has a separate clonezilla installation for reimaging and a grub option to restore the main partition from the master image on ctvr-gateway (admin password needed).

2.3.2.2 Incorporation of GPS and GUI functionality

Work has been carried to create a GUI for IRIS. The purpose of this GUI is to make the IRIS software radio easier to use and to provide a means of better presenting the functionality and metrics of a radio in a demo setting. The work on this is ongoing and is not yet available online. Work on the incorporation of GPS functionality is still ongoing also.

2.3.3 LTE 2.1 GHz front-ends

The LTE testbed has been extended with several 2.1GHz front-ends for the eNB and UEs. This extension was necessary, because the licenses for 2.6GHz areexpected to be withdrawn by the respective owners in near future as the commercial exploitation of these frequencies progresses in Germany.

2.3.4 Towards UWB and 3G Femtocells

For a testbed or federation to become and remain sustainable, one of the crucial criteria is “technological relevance”, meaning that an experimentation facility should offer those technologies that are of interest to experimenters. The different CREW facilities already offer access to a wide range of wireless technologies and software defined radios. Nevertheless, the IBBT w-iLab.t has studied the possibilities of extending the offering towards UWB and 3G Femtocells:

- There has been a demand for the availability of UWB devices, for use in positioning experiments. Rather than using such devices only for positioning for a single project, the idea is to install these devices in the IBBT w-iLab.t. The devices that are likely to be acquired are shown in Figure 14 below.

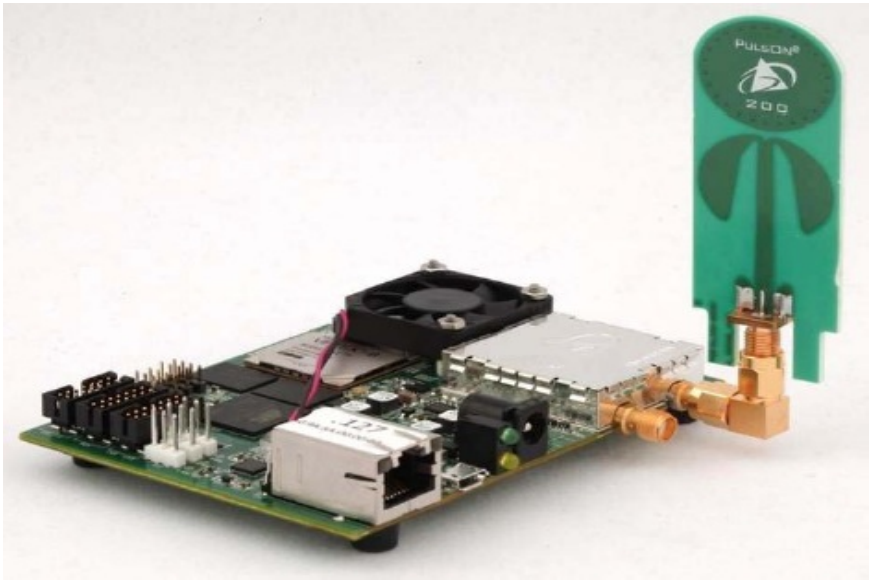

Figure 14: P400 RCM module

As the P400 RCM modules can not only be used for localization but also support 2-way data transport at a physical layer data rate of 158kbps, they can also be used to form sensor networks. As such, these devices could be used as interferers to create more realistic network environments, but could also be used as devices under test in UWB sensor networks. Note that while the equipment is relevant for CREW, this hardware is not bought using CREW budget. Parallel funding from Ghent University has been obtained for purchasing UWB devices.

- The interest of SMEs and companies in 3G/4G Femtocells seems to be increasing. Therefore, recently IBBT has started to explore the possibility to acquire Femtocells and Femtocell software. The challenge is not necessarily to buy Femtocells, but to acquire femtocells that allow experimenters to make the necessary changes. In addition, licenses need to be acquired. There is an ongoing discussion to acquire software and Femtocells via a European research institute. However, no concrete results can be shared at this moment.

3 Demand-driven extensions derived from external experiments

In this section we describe the extensions and actions that were necessary to support the experiments carried out by the new project partners who participated in CREW via the first open call (WP7). However, note that due to the late start of the open call partners, some activities anticipated in demand-driven extensions will occur in year three.

3.1 Support for the Durham experiment

University of Durham (UDUR) has been supported with the actions described in this section.

3.1.1 Support for the channel sounding measurements in the aircraft cabin environment

An aircraft cabin with its metallic structural components and outside shell is assumed to be a special environment for electromagnetic wave propagation. On the one hand, the cabin can be seen as a kind of wave guide for the fields, and on the other hand the great number of reflections will lead to distinctive frequency selective and fast fading effects. The Airbus A340 cabin mock-up that has been set up on the premises of EADS Innovation Works enables CREW partners to conduct their wireless experiments in the environment of an aircraft cabin. Although the cabin mock-up mainly consists of wooden parts, the coating of all structural components with a metallic foil leads to realistic wave propagation characteristics. The following figures show the inside and outside view of the mock-up.

The cabin environment is a good test case for the channel sounding activities of the University of Durham (UDUR). Sounding the aircraft cabin channel would also help to interpret the results of the internal experiments (cf. D6.2) performed in the aircraft cabin, by knowing the scattering function of the wireless channel.

The channel sounding activities will be supported by providing University of Durham all required information about the cabin mock up and by giving them access to the facilities to perform their measurements. All results and knowledge obtained during the channel sounding campaign will be transferred and made available for the CREW partners and the internal experiments.

3.1.2 Support for the channel sounder measurements at TUB

Prof. Salous of University Durham (UDUR) visited TUB during 11th – 13th April 2012 to discuss the envisioned measurements to be conducted at TUB. During this initial meeting we identified the main challenges in performing channel sounding experiments in the TUB building. The main goal of the experiment is to identify the dynamic multipath channel parameters including angle of arrival information and to understand the benefits of directional sensing and generate a suitable multipath channel model for a dynamic (movement of people) indoor environment.

To perform the measurements the indoor localization of the channel sounder is necessary. The device itself is too big to be carried by the mobile robot of the TWIST testbed (TWISTbot), which can roam

through the building by itself. It was decided to use the robot localization mechanisms and operate the trolley with channel sounder manually.

We have discussed possible signals that can be used for measurement at TUB as well as in an anechoic chamber in Durham and in IBBT and decided to use the signals artificially generated by Simulink model and played by signal generator. We will use common signal types specific to 2.4GHz ISM band like WLAN, Bluetooth and IEEE802.15.4. With this method it is be possible to generate exactly the same signal in Durham with their signal generator and at TUB. We have already exchanged the Simulink models and code necessary for signal generation. It is easy to reuse and replay such signals and decuples the useful signal data from the hardware.

3.1.3 Support for the channel sounder measurements at IBBT

Similar to the measurements that are planned at TUB, a measurement session is also planned in the pseudo-shielded environment of the IBBT w-iLab.t testbed (Zwijnaarde location). Due to construction works in the Zwijnaarde testbeds where additional air vents were installed (thus altering the propagation behaviour of the environment), the measurements have been delayed to Fall 2012. The results of the measurement session at IBBT will be included in the deliverables of open call partner Durham.

3.1.4 Support for the IMEC sensing agent

The IMEC sensing engine, both the version using the SCALIDO RFIC [5] and the version using the WARP[19] radio board, is part of the experiment UDUR experiment in the anechoic chamber. To allow for easy evaluation of the measurement results at the moment the experiment takes place, raw output samples of the sensing engine have been provided to UDUR in advance. To execute the actual measurement we foresee to use the existing API, which already has a mode available to store time-domain samples. If required for the experiment it could be possible to utilize the DDR RAM on the IMEC sensing engine board to obtain longer signal traces, which would require a software and firmware update of the sensing engine.

3.1.5 TCD support in anechoic chamber experiments

TCD will take part in the anechoic chamber experiment in UDUR. USRP equipment from the IRIS testbed will be used to perform simple energy detection, which will then be combined with similar readings from other devices in post processing. The CREW common data format will be used for the output data.

3.2 Support for the TUIL experiment

For the TUIL experiment, a gradual approach was selected to couple the IMEC Sensing Engine to the hardware from TUIL. In a first step, the software API for the Sensing Engine was adapted to calculate a power number for the selected channel. Starting from a capture of that part of the spectrum containing the selected channel, a 128-point FFT is performed. The bins belonging to the selected channel are then used to calculate a single power number, which is made available for readout. Apart from implementing the algorithm to calculate the power, support has been given on how to install the hardware, how to use the custom-made software packages, and some changes to the API have been made to accommodate the needs of the users. Also, a tutorial has been written on how to configure and use the Sensing Engine for the TUIL experiment.[20]

In the second step, the FPGA design on the platform was extended to be able to connect the Sensing Engine on the hardware level. A special interrupt handling routine has been implemented, which

[19] "Rice University WARP Project." Online available: http://warp.rice.edu

[20] http://www.crew-project.eu/sites/default/files/SensingEngine_UserManual.pdf

couples the output from the Digital Frontend for Sensing (DIFFS) chip to a General Purpose IO-pin on the Sensing Engine platform.

In the third step, a custom made firmware design has been created to perform the power calculations on the DIFFS chip, this is described in more detail in section 3.2.1. This enables much faster signal processing than the rather slow calculations in software on the host PC. The new firmware was tested in simulation and new filter configurations were calculated specially for the demands of the experiment. As mentioned before, this was expanded with an interrupt controller to output the result of each run of the experiment. After verifying the new firmware in simulation, the software API was once again modified to support the new firmware and was tested on the hardware. A verification test scheme was designed to determine the real-life behavior of the new test platform. After the implementation and test phase, the new API package was transferred together with instructions on how to use the new version of the Sensing Engine. Afterwards, support was given to finetune the Sensing Engine to the needs of the users and for the execution of the experiments.

3.2.1 Sensing Engine firmware upgrade

A part of the sensing algorithm implemented in the project is detection of energy for 2 and 4 MHz wide signals transmitted by the USRPs from TUIL. This functionality is implemented on the DIFFS frontend. The implementation details are explained in the following sections.

3.2.1.1 Hardware architecture

The digital frontend (DFE) is a component that fits in the radio receiver between the analog frontend and the basebandprocessing unit. It enables both synchronization and sensing features. For the implementation of the sensing algorithm, only a subset of the components of DIFFS is used. The platform overview in the next subsections is focusing on those components. Figure 15 Figure 15depicts the high-level block diagram of the device. The analog frontend, a WARP radio board for this experiment, is connected to the top input, and received data after synchronization is sent to the baseband engine from the buffer. The sensing-specific sub-blocks are: the AGRAC, the compensation block, the flexible filter branch and the SensePro.

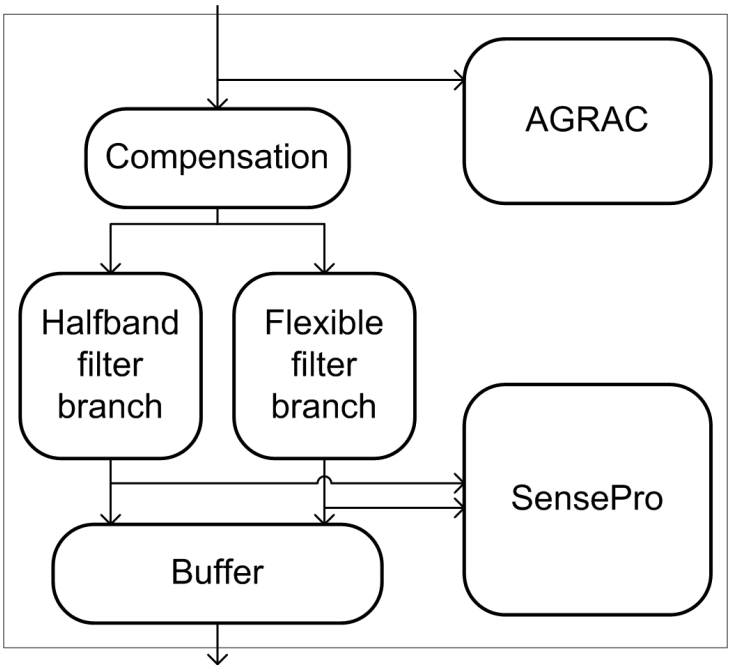

Figure 15: DFE block diagram

3.2.1.2 Top-level control and frontend interface (AGRAC)

The Automatic Gain and Resource Activity Controller (AGRAC) controls the DFE. It serves two tasks: determining the gain settings of the analog frontend upon reception of a radio signal, and controlling the enable signals of the other sub-blocks to implement a hierarchical wake-up mechanism. This principle of hierarchical wake-up exploits the fact that only the blocks consuming minimum power are always active. For sensing applications, the DFE can work in multiple stages: if no power is detected by the simple power detection in the AGRAC, more powerful signal processing algorithms on the SensePro are activated. When the AGRAC detects power, the channel is found to be occupied and no further sensing is needed. The AGRAC is implemented as an 8-bit microcontroller that is compatible with an industry-standard C toolflow to ease programming. The core of the processor runs at the speed of the incoming samples and contains a peripheral that measures the received signal power and the DC offset.

3.2.1.3 Compensation and filtering

The incoming samples are first sent through a block that compensates for frontend non-idealities. In this block, the remaining DC offset is digitally removed and I/Q imbalance compensation is applied. Subsequently, two parallel filter branches can operate on the data. One filter branch is a power-optimized set of half-band filters, the other branch is a fully programmable filter branch that supports down-conversion, band selection, and non-integer rate conversion (Figure 16). All filter blocks are implemented in full precision, and a quantization selection block after each stage allows the algorithm designer to select the most relevant bits from the output of the filter. The filter block that is used for implementing the sensing algorithm for the TUIL experiment is the flexible filter branch.

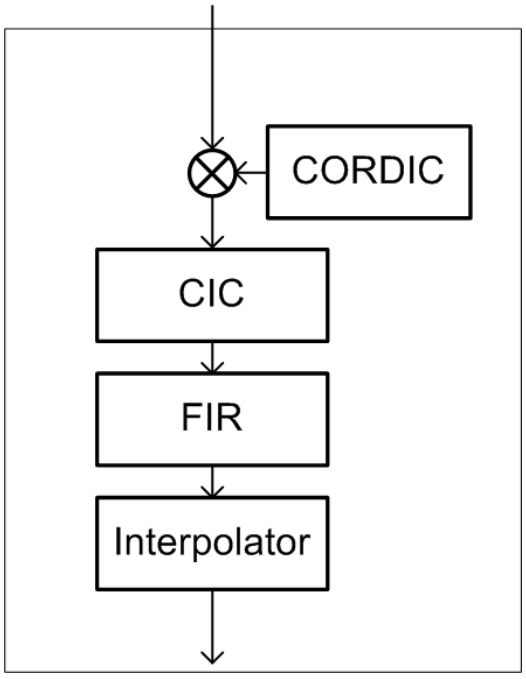

Figure 16: Flexible filter branch block diagram

For implementing the sensing algorithm, the following parameters are used for the flexible filter branch:

- 4 MHz mode
    - CORDIC offset: -2/-1/0/1/2 MHz depending on the requested channel center frequency. The value is dependant from the sampling frequency, which is 40 MHz in this case. For example, an offset of -2 MHz corresponds to a CORDIC angle of -2/40 or 38/40. Therefore the hexadecimal equivalent of 0.95 has to be programmed, which is 0xF3333333 (~ 0.9499999999534).

- CIC filter: the number of stages is 2, the number of delay elements is 1 and the decimation factor is 4.
- FIR filter: the filter settings consist of 21 tabs that are to be programmed.
- Resampler: the resampler or interpolator is not used for this mode.

- 2 MHz mode
  - CORDIC offset: -2/-1/0/1/2 MHz depending on the requested channel center frequency (see above).
  - CIC filter: the number of stages is 3, the number of delay elements is 1 and the decimation factor is 4.
  - FIR filter: the filter settings consist of 21 tabs that are to be programmed.
  - Resampler: the resampler or interpolator is not used for this mode.

3.2.1.4 Synchronization and Sensing Engine

After passing through the filters, the data enters the SensePro synchronization and sensing engine. This is an ASIP that contains a 32-way SIMD processor with scalar slot and three accelerator cores (Figure 3). The SIMD processor is used for all the flexible operations that are required for different sensing algorithms, such as min/max, averaging and thresholding. The scalar slot is used for loop control. The accelerator cores handle the tasks that do not map efficiently on the SIMD, such as 128-point Fourier transform, vector rotation and correlation operations. The correlator is specifically used for synchronization, as this function needs to happen in a low power fashion for an efficient hierarchical wake-up of the radio. The SIMD processor data path supports complex data samples and every SIMD slot is 2x16 bits wide. All vector instructions are targeted towards synchronization and sensing specific functionality so that the various algorithms can be mapped efficiently. As the processor only contains a scalar and a SIMD slot, firmware development for the SensePro is fairly straightforward. Programming is done in assembly. The SensePro runs at the incoming sample speed and can clock-gate itself when it is waiting for data from the correlator that generates the input vectors from the input samples. No PLL is used, to reduce the power consumption.

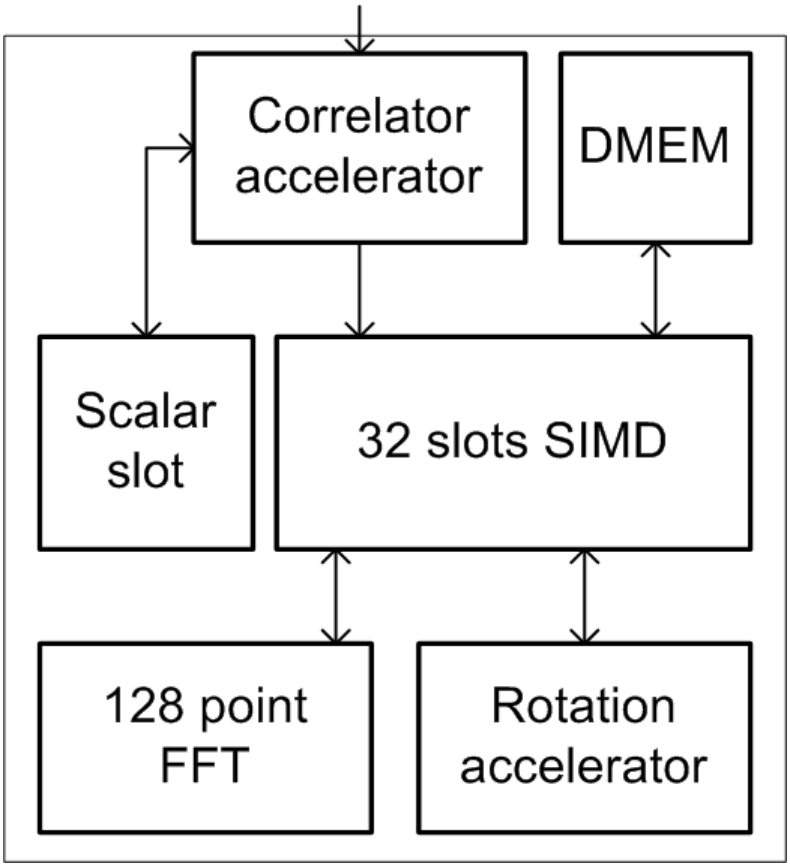

Figure 17: SensePro block diagram

3.2.1.5 Software implementation

The DFE is a configurable part. Next to the control parameters for the hardware configuration of the filters, an important part of the functionality if determined by the software that is running on both programmable ASIPs: the AGRAC and the SensePro. This section details the software that is mapped on both cores to implement the sensing functionality.

As the sensing algorithm requires detailed information on the received signal power, the limited power detection hardware on the AGRAC does not suffice. Hence the deployed strategy is to use the AGRAC for basic frontend control, and then to activate the SensePro to calculate the required signal properties.

3.2.1.6 AGRAC

The AGRAC is running a straightforward algorithm: after reset, it will program the analog frontend to a fixed gain setting, and then it activates the compensation block, the filters, and SensePro. From that point on the SensePro is analyzing the incoming (filtered) samples.

3.2.1.7 SensePro

The SensePro is configured to analyze the incoming samples as follows:

- The correlator core is programmed in pass-through mode, as the calculations will be performed on the vector slot
- The absolute value of the vector values is calculated on a slot-per-slot basis.
- The maximum result value is determined.

- If the maximum value is bigger than the programmed threshold, the interrupt signal of the SensePro is asserted. When the value is smaller than the threshold, the interrupt signal is de-asserted.

3.2.2 Measurement results

The sensing engine was connected to a wide band signal generator to verify the operation of the implementation described in the previous sections. The signal generator was programmed to generate an OFDM signal with a bandwidth of 2 MHz. The power level of the RF signal was varied from -100 to -20 dBm and 1000 measurements were done for a large number of thresholds between 0 and 3000. For each combination of input power and threshold we compute the number of measurement values that exceed the programmed threshold. The results are shown in Figure 18, the X-axis shows the input power level in dBm, the Y-axis contains the threshold value and the curves indicate the percentage (25% - blue / 50% - red / 75% black) of measurement results that exceed the programmed threshold value.

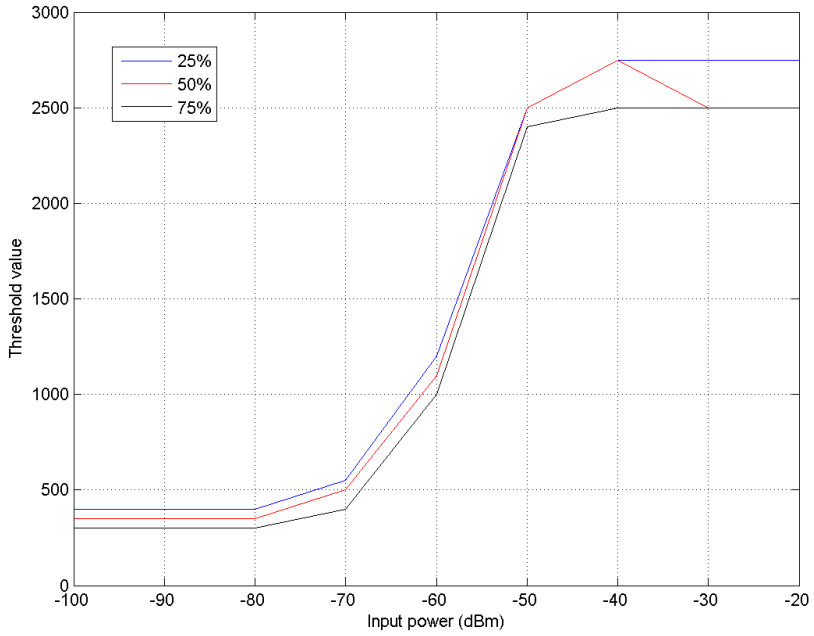

Figure 18: detection performance of sensing engine

All three curves, 25, 50 and 75%, show a very similar trend and are very close together, therefore we can conclude that we have on/off behaviour in the sensing engine which is the wanted behaviour for this experiment. This curve can be used as reference by TUIL to select the correct threshold setting

3.3 Support for the TECNALIA experiment

3.3.1 TCS Transceiver Facility API

Thales Communications & Security (TCS) interaction with TECNALIA was limited to provide a support document for the Transceiver Facility API software, given in CREW deliverable D3.2 section 5.2, as well as the implementation itself of that programming interface (Transceiver Facility API reference code) for the widely available and used Ettus Research USRP2[21] board environment.

[21] Ettus Research, https://www.ettus.com/

From these 2 materials, TECNALIA succeeded in integrated Thales Transceiver Facility API [3] software for running its CREW experiment.

3.3.2 Support for the IRIS Platform

Thus far, the support provided by TCD has taken the form of providing instruction on the installation, set up and development of IRIS to accommodate the first stage of the TECNALIA experimental implementation. This first implementation was performed in the TECNALIA testbed and made use of Iris as a means of implementing the algorithm for combining sensing information received by multiple USRPs operating using the TCS Transceiver Facility API [6]

The following stages will involve implementations within the IRIS testbed in Dublin and will require strong collaboration and involvement, as well as use of IRIS, the IRIS testbed nodes and other IRIS testbed equipment. Details of the implementation plans can be found in D7.3.1. At the upcoming CREW meeting in Durham (October 2012) the second stage of implementation (node migration from TECNALIA testbed to TCD) will be discussed.

3.4 Support for open call experimentation through implementation of a MAC/Network Layer in Iris

As part of the "General support/extension of the Iris functionality" work of the Iris testbed and in order to facilitate cognitive network based experimentation within the Iris testbed, work on the development and implementation of a MAC-Network layer for the Iris SDR has been performed. There has been large demand for, previously unsupported, network level experimentation within the Iris testbed, support for which has been needed by both the open call partners (TUIL and TECNALIA) and numerous external users of the Iris testbed facilities in the CREW federation. This work expands the CREW federation experimental cognitive network functionalities greatly.

Work is ongoing on the implementation of a carrier-sensing-based medium access control (MAC) protocol. The following is an abstract from a work recently submitted to Transaction on Vehicular Technology which outlines the work.

Abstract: *Implementation of carrier-sensing-based medium access control (MAC) protocols on inexpensive reconfigurable radio platforms has proven challenging due to long and unpredictable delays associated with both signal processing on a general purpose processor (GPP) and the interface between the RF front-end and the GPP. In this paper, we develop a split-functionality implementation of a random access carrier sensing MAC, in which some of the functions reside on an FPGA and others reside in the GPP. We provide an FPGA-based implementation of a carrier sensing block and develop two versions of a CSMA MAC protocol based upon this block.*

We experimentally test the performance of the resulting protocols compared to previously-developed, Aloha-based MAC protocols in a multihop environment, demonstrating improvements in both throughput and required frame retransmissions. We crossvalidate these results with a network simulator with modules modified to reflect the mean and variance of delays measured in components of the real software-defined radio system.

4 Conclusions

In this document we have described the initial demand-driven extensions of the CREW federation. We explained how the federated CREW test facilities have been extended with a first set of new functionality which has been defined in a demand-driven and open way based on the gaps identified from feedback CREW core members and their involvement in the FIRE community as well as external experimenters who participated in WP7 via the first open call.

To this end we have adopted the *Connectivity Brokerage* (CB) [1] architecture in CREW in order to allow CREW experimenters to integrate their mechanisms-under-test in a well-specified CR framework and facilitate experimentation by enabling reuse of CR software components and concepts. However, we identified a set of limitations that prevented us from applying the CB framework directly, in particular the lack of specification for the communication among CB objects. An important extension carried out in WP5 was therefore the extension of the CB framework and its proof of concept via adoption in CREW usage scenarios. In addition, the CREW involvement in the FIRE community resulted in establishing a collaboration between CREW and the GENI initiative in view of the joint development of a common cognitive radio language.

Furthermore, a set of new hardware was identified as a valuable extension to the CREW federation. These extensions involved integration of low-cost USB spectrum sensing devices into the TWIST testbed, extension of the TUD testbed with 10 additional nodes and extension of the LTE testbed of TUD with with 2.1GHz front-ends.

Finally, we described the extensions and actions necessary to support the experiments carried out by the new project partners who participated in CREW via the first open call (WP7): for University of Durham (UDUR) this involved the support for the channel sounding measurements in the aircraft cabin environment, as well as preparing the channel sounder experiments at the TUB & IBBT testbeds and the preparation of the experiments in the anechoic chamber at UDUR. The TUIL experiments were facilitated by supporting adaptation of the API of the IMEC Sensing Engine and extending tutorials. Finally, the experiments conducted by TECNALIA involved writing a support document for the Transceiver Facility API software as well as joint preparation of the first stage of the TECNALIA experimental that involves the IRIS platform.

5 References

[1] Rabaey, J., Wolisz, A., Ercan, A., Araujo, A., Burghardt, F., Mustafa, S., et al. (2010). *Connectivity Brokerage - Enabling Seamless Cooperation in Wireless Networks.*

[2] Parsa, S. B. (2010). Design and Implementation Guideline for the Connectivity Brokerage Distributed Repository (CBDR). Berkeley Wireless Research Center (BWRC). Available: https://bitbucket.org/aparsa/connectivitybroker.

[3] Joseph Polastre, R. S. (2005). Telos: enabling ultra-low power wireless research. IPSN '05: Proc. of the 4th international symposium on Information processing in sensor networks. Los Angeles, California, US.

[4] P. Levis, D. G.-H. (2005). T2: A Second Generation OS For Embedded Sensor. Telecommunication Networks Group, Technische Universitaet Berlin.

[5] M. Ingels, V. G. (2010). A 5mm2 40nm LP CMOS 0.1-to-3GHz multistandard transceiver. in 2010 IEEE International Solid-State Circuits Conference ISSCC.

[6] E. Nicollet, S. P. (2009). Transceiver Facility Specification. *"SDRF-08-S-0008-V1_0_0_Transceiver_Facility_Specification.pdf", "http://groups.winnforum.org/p/cm/ld/fid=85".* Wireless Innovation Forum.

6 Appendix

6.1 Support Letter from UC Berkeley

UNIVERSITY OF CALIFORNIA AT BERKELEY

BERKELEY · DAVIS · IRVINE · LOS ANGELES · RIVERSIDE · SAN DIEGO · SAN FRANCISCO SANTA BARBARA · SANTA CRUZ

Berkeley Wireless Research Center (BWRC)
2108 Allston Way, Suite 200
Berkeley, CA 94704-1698

E-MAIL jan@eecs.berkeley.EDU
FAX: (510) 883-0170
OFFICE: (510) 666-3102

September 19, 2012

TO ALL CONCERNED:

Hereby I would like to express the interest of the faculty at the Berkeley Wireless Research Center (BWRC) to actively collaborate with the CREW project in a number of ways. BWRC is a 13-year old research center, part of the Electrical Engineering and Computer Sciences Department of the University of California at Berkeley, focusing on research in disruptive next-generation wireless technologies.

Our intention is to collaborate with CREW along a number of lines:

- BWRC would cooperate with CREW researchers towards the prototyping of a concept called "Connectivity Brokerage" (a concept pioneered at BWRC), which enables a dynamic trade-off over heterogeneous wireless networks based on demand and availability.
- For this activity and others, BWRC would make use of the testbed facilities developed under CREW (especially the ones at Ghent University and the Technical University of Berlin). One particular area of interest is the study of how distributed interference discovery and mitigation can help to increased capacity and energy of heterogeneous wireless networks.

For these collaborative activities, BWRC would contribute its own financial resources. BWRC is a research consortium, funded by most of the major industrial players in the field (for a complete list of members, please consult the BWRC website at http://bwrc.eecs.berkeley.edu). In addition, funding for this particular project is provided by the FCRP Multiscale System Research Center (MuSyC), jointly funded by the semiconductor industry and the department of Defense.

It is our belief that large-scale and scalable cognitive testbeds, such as provided by the CREW project, are essential for the evaluation of the effectiveness of innovative wireless system technologies. We are therefore excited to be a part of this effort.

Sincerely,

Jan M. Rabaey
Donald O. Pedersen Distinguished Professor
Director Multiscale Systems Center (MuSyC) and Berkeley Ubiquitous Swarm Lab
Scientific Co-director Berkeley Wireless Research Center (BWRC)
University of California at Berkeley

6.2 Support Letter from WINLAB, Rutgers University

RUTGERS
WINLAB | Wireless Information Network Laboratory

WINLAB
Technology Centre of New Jersey
Rutgers, The State University of New Jersey
671 US Route 1
North Brunswick, NJ 08902-3390

732-932-6857 Ext. 640
Fax: 732-932-6882
seskar@winlab.rutgers.edu
www.winlab.rutgers.edu

To: Whom it May Concern
Date: March 16, 2012

RE: ORBIT/GENI collaboration with CREW project

This letter will confirm the intention of WINLAB, Rutgers University to collaborate with CREW on the topic of testbed experiment management and control. The following coordination topics have been identified with CREW team:

1.) **Experimentation support**: facilitating experiment exchange and allowing CREW resource use for GENI community as well as CREW user community use of GENI/ORBIT resources.

2.) **Federation support**: harmonization of existing interfaces to support experiments which concurrently use resources from both testbeds.

3.) **Joint development**: Both sites are actively using OMF framework for testbed management. OMF includes constantly evolving Experiment Description Language (OEDL) that enables efficient definition and configuration of required resources required as well as state-machine based experiment task description. We plan to work with the CREW project on the development of a common CR language to facilitate experimentation with heterogeneous CR platforms deployed in testbeds in both institutions including: USRP & USRP2 with GNURadio or Iris, IMEC sensing agent, WARP, GENI CRKit and other programmable platforms.

In addition, we are also committed to preparation of join demos for various GENI Engineering Conference (GEC) and equivalent European Project Conferences.

The ORBIT/GENI team at WINLAB looks forward to cooperation with CREW's Service Delivery and Testbed Framework project. Please feel free to contact me at seskar@winlab.rutgers.edu if you require any further clarifications.

Sincerely,

Ivan Seskar
Associate Director
WINLAB, Rutgers University
Technology Cetner of NJ
671 Rt. 1 South
North Brunswick, NJ 08902
Tel: +1 (732) 932-6857 ext. 640
URL: http://www.winlab.rutgers.edu