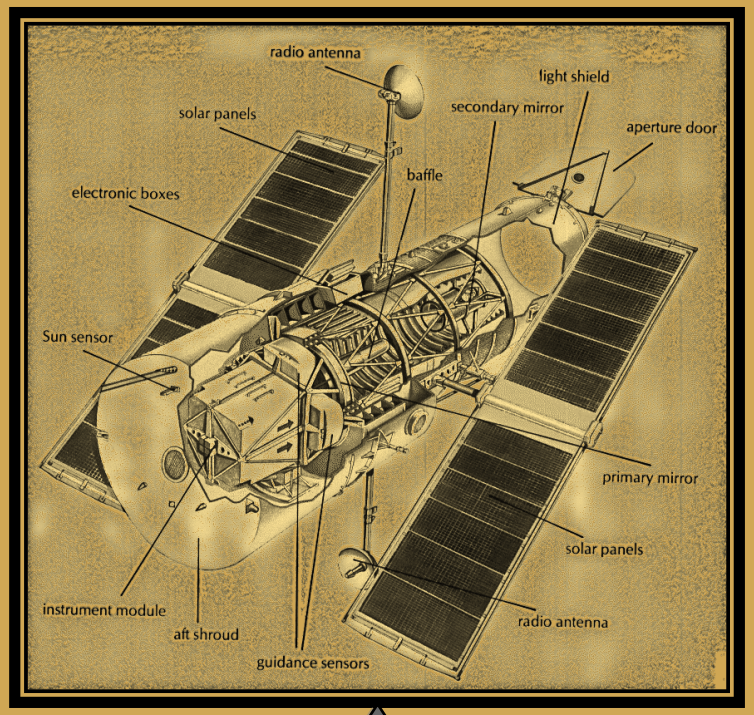

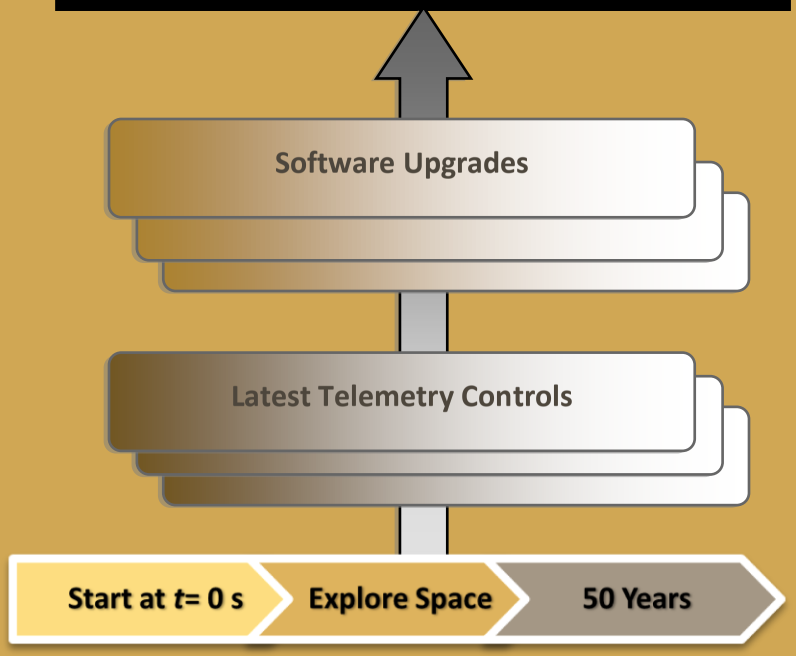

The Universe Explorer

Architecture Design

BLEKINGE TEKNISKA HÖGSKOLA, Sweden

Philip Baback Alipour, Oleg Kustov, Hiva Allahyari

October 15, 2010

The Universe Explorer

Architecture Design

Abstract

In this document, we have elaborated on the key terms and designed the Universe Explorer (UE) system with relevant components, using IS2000 fictional system on the signal processing, communication and data acquisition parts, as a standard to this work. Nevertheless, a customized hardware-software engineering notation to our design from different viewpoints, Conceptual, Modular, Execution and Code, is also visible. The customized notations were used for our main *global analysis topologies* due to incorporating latest technological standards and beyond, within the systems' specifications domain. The topologies represent the foundation of our work in terms of both, hardware and software of the UE system to maintain communication between Space and Earth, since this is the primary focus of the software architecture. We have for our entity managers and controllers, also proposed novel solutions in our architecture, such as carbon nano-tubes (CNTs) as a hardware and sensory component, for UE shield and thermodynamics. A new invention-model of a parallel time-varying CNT-microchip, is also proposed, satisfying UE's 50-year or longer mission, addressing issues concerning *information latency*, *parallel processing*, and *data integrity* (avoiding any data loss) relative to a *lossless data compression algorithm* (LDC), on its databases, security, etc., transmitted from Space to Earth. This LDC, with its predictably-fixed compression ratios, fulfills the software-hardware strategies outlined on data processing constraints, such as temporal and spatial limits against random LDCs, thus making the software application to deliver binary data, real-time, rather than random time, from an execution viewpoint. The current work is novel in its software and hardware quality framework, and to this account, all views are discussed with relevant examples to the architecture. Agent-based trainable algorithms evolving the instruction set architecture (ISA) on entities inclusive of a 2*n*-core CPU soft-switch semaphore solution, throughout the UE mission, are also discussed in this document.

Keywords: Processors, data compression, binary database, entities, connectors, roles, software quality, UE satellite.

Table of Contents

1. UE System Context and Introduction

Construction, *instrumentation*, and *control* of a Satellite System for a Universe Explorer (UE) Mission or spacecraft project, reaching beyond the limits of our galaxy to the far ends of the universe, can be divided into four main categories: *operational UE, all time/real-time communication UE, tolerance-related UE, and application upgrade-related UE*.

The UE should in its system specification include supportive parts or components for a *healthy communication* between Earth and its control units from the point of data gathering in Space. A "healthy communication" here, means data to be transmitted as efficient as possible, without errors or scrambled messages delivered to its point of destination on Earth. Therefore, the design should also be fault-tolerant, equipped with *checkpoints* and real-time hardware/software management issues. The system must also exclude irrelevant data on the spot as defined by the system specification manual, to avoid any e.g. memory overruns when receiving raw data (primary) no matter how packetized or compressed in terms of information technology utilized onboard for a range of frequencies. We generally must maintain *meaningful data* as secondary high-level information after certain primary data manipulations onboard the satellite, when received on Earth for further processing and thus analysis.

When we mean meaningful data, is the relevance of information within the Knowledge Engineering context that is valuable to the UE remote users on Earth which decipher it as imperative data to their research and global presentations. It could be classed as e.g., whether forecast of some planet, bio-

signatures indicating life, chemical elements useful to a human environment, photographic-based geological data, etc. Such data could be in form of digitized and analogue signals as voice patterns, imagery type, etc. So, the UE system must also be equipped with all sorts of subsystems to receive and deliver the right information whereas this information is defined by the designers of the UE knowledge based system useful for a particular mission. For example, one could specialize the regular tasks carried out by UE program routines to check for only life signs for a certain period of time e.g., when close to planet Mars, and then shift back to its universal routines for real-time reports. These instances are most likely to happen when data is being analyzed on Earth, once detecting some important sign of e.g., life, commanding the satellite to head back to its previous spot to verify some of that information for us observing events on Earth.

In this chapter, we shall introduce the UE system in its context and demarcate the vital global elements of its functional components on design activities. From there, inclusive of examples where safe vs. critical scenarios could happen in its 50 year mission in space, we discuss and analyze our engineering system (global analysis) in sign of delivering the conceptual, module and execution architecture views in Chapters 2, 3, 4 and 5 respectively. The presentation and structure of our current report obeys the way where applied architectural contributions are made in Hofmeiester *et al.* course book [1].

We first specify our functions map in form of a full-system description and its overall specification as a preamble to UE *conceptual*, *module* and *execution* viewpoints, relative to its *quality requirements*. We further imply the latest hardware and software components relevant to extreme (harsh) environments where temperature, magnetic fields, lack of sunlight causing extreme subzero conditions, disrupting data I/O products between UE system units, even affecting hardware to function poorly or shutting down indefinitely. This obviously, for extreme conditions requires a backup system intact with the unit parallel to actuators for its navigation system. The backup system includes redundant data paths for replacing a damaged circuit with a healthy one, DB reserved partitions for control units in case of failures, and thus a memory management system for efficient processing of input signals (*data acquisition* by onboard sensors, signal conditioning circuitry and analog-to-digital converters, or see, Chap. 8 [1]).

The motive for our system base in its design, description and topology, was a simple client-server model, where we contemplated the *incoming signals* from space to the UE system, as the “source client”, and the server part, the UE itself, receiving more signals when necessary from Earth as the “command” or “sink client” connected to the satellite.

The main hardware and software topology for a complete system description is given in Figs. 1-3. Fig. 1 represents the bottom plan (Pl. A): the base of the UE satellite system or its anatomic foundation, core of operations, mechanics, thermodynamics and network. For its mechanical state avoiding obstacles and problematic issues in space or by contrast, maneuvering the system to specific geographical points in space, came about the knowhow robotic and machinery systems work in our real world environments. Of course, the main assumption to this is realizing that space is a vacuum and objects floating in it are classed as planetary, stellar, and other forms of bodies regardless of weight constraints, physical dimensions and behavior to act free in their own account until an interacting force influences them to fall into certain physical restrictions. So, we initially staged our UE to have been launched into space and the basics of weight analysis, atmospheric pressure constraints such as friction between its body (bus) and air, have been dealt with on Earth before its space exploration chronology or its 0s to 50 year mission. We also assume that the costs above $0.5 billion according to NASA’s official reports [4] have been covered for this launch and the UE system, of course, with multiple backup systems have been graciously considered by its manufacturers. To this account, the main focus is to maintain the most efficient way for a 50 year timeline successful mission of the unit working in space, transmitting data to Earth and vice versa. The latter, however, is mostly of a critical condition handling type or a command module type giving further instructions to the satellite for a particular task based on data collection priority, management, processing and communication.

1.1. The UE Satellite Class

The class of this satellite is specialized for travelling to the far ends of the universe on a scientific mission, whilst equipped with functions performing tasks identical to those tasks performed by other satellite classes, according to [3]. Nowadays, satellite technology comes with all shapes and sizes, and in general, are classed as follows: Weather satellites; communications satellites; broadcast satellites; navigational satellites; rescue satellites; Earth observation satellites; Military satellites and amongst them, the most relevant class to the UE system, would be Scientific satellites: They perform a variety of scientific missions. The Hubble Space Telescope is the most famous scientific satellite, but there are many others looking at everything from sun spots to gamma rays [3]. Most satellites are orbit based, however, noncommercial scientific satellite that travel between orbits through e.g. slingshots and other mission for surveying planets must have unique capabilities for long missions when attaining at some alien-like orbit level at some planetary level in space. A good example is Cassini-Huygens satellite or spacecraft probe [27].

The UE satellite, of course, in its generalized tasks criteria, must be able to perform some of the tasks outlined above, such as: *Weather, communications, geological changes (observation), photographic, etc*. In its specialized class, however, it is simply *scientific* looking at everything e.g. alien life forms throughout its mission. In other words, the UE must satisfy a combination of proficient abilities conducted by other satellite, prior to its expertise as outlined in its system specification for its scientific mission, once lunched from the Earth's atmosphere to space.

1.2. System Description and Specification

The UE satellite, is a very customized, a very pinpointed designated system for specialized tasks, and is unlike the heaviest communications satellite with a weight of 6.5 tons that was successfully launched to satisfy a commercially-oriented, human-consumer grade media technology on Earth [4]. It is "scientific", noncommercial, and differs from other satellite custom product lines, with a specific purpose to explore the ends of our universe.

1.2.1. Components for Hardware-Software Description and Specification:

The Building Blocks for the Conceptual View

The UE is there to *send us information* in aim of building our *knowledgebase system* on Earth. For this assignment, UE requires for its hardware/software units to filter out information highly-efficient with the latest technologies installed onboard.

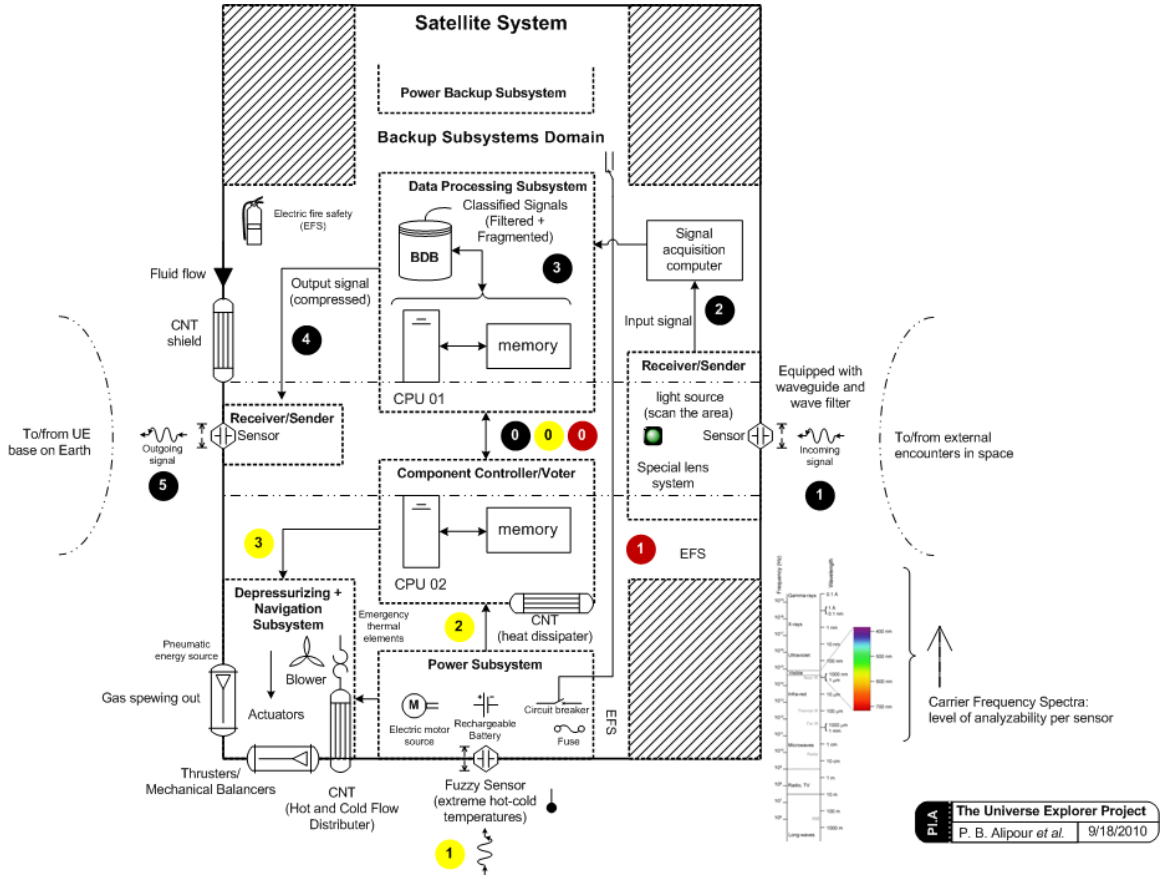

Fig. 1. The UE Satellite hardware and software topology in space, receives data, processes it and sends it to Earth. This plan also shows the supportive features of the satellite for successful data transmissions, actuations, power supply and backup systems onboard under critical and regular circumstance. The hashed areas are the identical version of the lower-left corner subsystem, satisfying UE's areal maneuverability air and temperature flow control.

We have shown this for at least two CPUs working in parallel for incoming raw data with manipulative features on certain levels of information (like data compression). Our choice of two CPUs, is motivated and rationalized in the evaluation section ahead, §1.2.2.

In continue, we even considered that our units will not weigh much in total, inclusive of their backups, up to 2*n*-CPUs or subsystems due to the incorporation of CNTs as a very light-weight solution. In Fig. 1, Plan A, as the lower-plan of the satellite (compare with the upper-plan, Fig. 2), mostly represents the vital components of the satellite's hardware: Actuators, CPUs and Memory with main processing and control unit paths within its exploration missionary context. Furthermore, it includes in its architecture solutions switches and shifters for such actuators and electronics respectively for emergency calls as its backup system or compensators. In addition, Fig. 1, partly includes a software solution in terms of a binary database (BDB), which is chiefly considered for processing, storage and data management issues like e.g.,

data retrieval, filtering, communication choice of shortest paths for controllers, compression, etc., *by referring to the right binary index indeed*.[1]

As we can see in Fig. 1, we have defined the successive steps in terms of *regular* with a **shaded black** background, *critical* with a **light yellow** background, and *highly-critical* with a **red background**. We specify these steps as follows:

Fig. 1 Regular steps (*routines*):

0) The UE is equipped with two CPUs assuming that both could even be manufactured with latest multi-core nanotechnology (see explanations after steps): CPU 01 is dedicated to data processing and DB management issues before sending the resultant data to Earth. CPU 02, however, is dedicated to manage and monitor all hardware components, their variances in performance, including instabilities within the chamber that might occur throughout this mission. The CPUs with their memories are presented in terms of 'von Neumann architecture' denoting standard instructions set and communication scheme, relative to this parallel processing between units over all of the subsequent steps of UE.

1) The UE receives/picks up a remitted signal from something/body/object in space via its sensor. The remission is a pre-condition to the current sensory event either projected to that thing from the UE satellite (the light component) installed on the telescopic level (the canalized dotted compartment above or inclusive of the sensor)

2) The input signal after being filtered and preprocessed via the signal acquisition computer, subsequent to the sensor, is delivered for processing to the data processing subsystem.

3) Here, data is processed and also classified (what type of data as e.g. sound type, imagery, etc.) and thus the irrelevant parts of the signal that occupy void, space, static or extraneous data are removed or excluded on the database. The signals are processed in *n*-equipartitioned forms or parallel incoming flows (*k* imagery sensors mounted on board the satellite) receiving *packets* of e.g., *k*×100 kBps, which is quite visible in imagery data types for image reconstruction purposes on Earth (compare to [56]). *Along this step*, a lossless data compression is applied to make it really efficient and easier data transmission between UE and Earth (further on this compression technology, refer to the paragraphs after these steps or **iii**). Therefore, in our Conceptual View, § 3.1.2, we simplify this applicable technique between the *data compression* criteria and *acquisition agents*, hence packetizing data is *encapsulated* on the signal acquisition portion of our system.

4) The output signal is a compressed signal ready for delivery to Earth at the speed of light or even alternative speed beyond this barrier speed limit proposed in §2.1.

5) Data is sent to Earth via the tail sensor which could await further instructions from Earth (an uplink) in case of software-hardware problems aboard the satellite.

Fig. 1 Critical steps (*exception handling routines*):

0) This pre-step reiterates "Step 0" from the regular steps.

1) The UE receives an e.g., extremely hot stellar object (like our Sun) via its Fuzzy Sensor. The fuzzy sensor is active at all times satisfying fuzzy states or multi-valued logic of the temperature according to its four-state detection ability e.g., extremely hot, extremely cold, somewhat hot and somewhat cold temperatures (see Fuzzy logic [33, 34])

---

[1] *Blob Computing* [53], relative to the lossless data compression (LDC) algorithm presented by Alipour and Ali in 2010 [6], or see exemplar **iii** below.

2) The power subsystem accumulates energy on a regular basis, from the second plan (upper-level) or wing sensors (Fig. 2) as they convert light energy to electricity (charge and electric current). The power system also reports fluctuations coming directly from the fuzzy sensor as an extra energy resource to the component controller/voter subsystem which ultimate triggers the control functions for exception handling routines.

3) **A-** While CPU 02 is monitoring such events since Step 0, it instructs through a voting state an actuator to operate in terms of running the blower component inclusive of the fluid flow in the *highly-light weighted, durable, flexible, resistant carbon nano-tube* (CNT) shield (see also Fig. 2). Before this, the air in the chamber is depressurized to create an air flow. This enables the controlling system to cool off the external and internal subsystems. The blower is activated only under extreme temperatures affecting the internal parts of the UE. **B-** In a different scenario, say, navigation, the actuator converts the electric signal to a thrusting power releasing the compressed gas into space to create a bodily reactive motion (see also **ii**). **C-** In another scenario, under intense gravitational fields causing extreme pressure on the UE body, the thrusters reactivated to escape from the field, and are relative to CNT *inflation protocols* (material expands or swells outward) to keep UE structural integrity intact on its hardware components indeed. This characteristic is defined within the limits of the CNT shield material.

Fig. 1 Highly-critical steps (*failure handling routines***):**

0) This pre-step reiterates "Step 0" from the regular steps.

1) The UE experiences some mechanical, hardware or circuit failure based on some too-late to-compensate situation. The system is equipped with relevant e.g., circuit-breakers, backup components (compensators) and backup paths within the backup subsystems domain, reroutes and switches from a faulty set of components or the failed circuitry to a new one (the blank area is occupied with these backup components). Under extreme situations where the circuit is on fire, the extinguisher shown in the system, which is highly symbolic, gets rid of the flame with a relevant chemical agent. However, the system is so-deigned in terms of containing such chemical agents stored in the wall of the UE system; say the CNT or the internal lower layers of the shield, and in such moment, the wall holes adjacent to the damaged area release the agent to extinguish fire. Other holes are sealed and would not allow further wastage of the agent to unnecessary healthy parts of the system. (Further feasible backup examples are given for the UE system in the following paragraphs.)

Exemplars and further descriptions/specifications of Fig. 1 components:

i. The backup battery cells located in the power subsystem are in two forms: one for instant use, which is rechargeable by switching to its neighboring cells when running low; the other, is for long term use. We suggest the use of carbon nano-tubes (CNT) paper batteries (CNT-PB) due to their size and physical characteristics operating beyond a typical industrial battery in use. They act as electrodes; allowing the storage devices to conduct electricity. The CNT battery, which functions as both a lithium-ion battery and a super-capacitor (or ultra-capacitor), can provide a long, steady power output comparable to a conventional battery, as well as a super-capacitor's quick burst of high energy—and while a conventional battery contains a number of separate components, the paper battery integrates all of the battery components in a single structure, making it more energy efficient [11]. It exhibits long life with little degradation over hundreds of thousands of charge cycles. Therefore, due to the capacitor's high number of charge-discharge cycles (millions or more compared to 200 to 1000 for most commercially available rechargeable batteries) it will last for the entire lifetime of most devices. Rechargeable batteries wear out

typically over a few years. The CNT-PB can help in conjunction with the prolongation of batteries lifetime by acting as a charge conditioner, storing energy from other sources for load balancing purposes and then using any excess energy to charge the batteries at a suitable time.

ii. For the UE navigation system, we considered a standard thrusting subsystem [5] equipped with compressed gas to spew out particles when decompressed into space, which is based on Newton's third law,[2] causes a reactive motion for the satellite's body. It gives a drivability function to the satellite, changing course from its current direction. In the long run, in case of running out on this particular energy source (of pneumonic type) , we also considered balancers that merely use a physical object to apply force to a corner point of the satellite's body. The actuating device obeys Newton's second law of motion $F = ma$, with an object with mass m undergoes an acceleration a, which outputs the opposite direction (Third law) of the force F, or $–F$, enabling the satellite to move in a particular direction. The output is equal in magnitude between the applied force F and exerted force $–F$, for a measurable momentum of the body in the right direction. This measurement is satisfied by the sensors installed onboard in the same area. Of course, this is commanded by the CPU 02 subsystem for this electric to mechanical conversion of energy when necessary.

iii. The lossless data compression (LDC) does not have to be on the basis of Shannon codeword standard [7], which might take some time in compressing data losslessly due to the random character/symbol repetition in data products between source and the units onboard [6]. In fact, a new LDC technology could be incorporated or utilized as one of the authors has already demonstrated for fixed compression ratios delivering double-efficient, quadruple-efficient, etc. data in a lossless manner. The logic or premise of the lossless compression model is called *fuzzy binary AND/OR compressor* (FBAR). This has been already introduced and evaluated on a smaller scale of data integration by Alipour and Ali, in 2010 [6], proving, no matter the amount of input data, using an ASCII-based translation table (**TT**) algorithmic component as a *static key*, gives 50% output, and using 2**TT**s in the program gives 75%, using 4**TT**s gives 87.5%, … so forth, as predictably-fixed compression ratios. In fact, a precise timing of data reception and thereby delivery between communication points is evident between Space and Earth under almost all DB circumstances. For one, the way data is allocated between the memory system and its DB application, would then be normalized in terms of its partitioning, thus resulting in 'predictable spatial and temporal limits' of satellite to Earth communication levels. Moreover, the delay point or any time allocation for compressing data is resolved by having the multiprocessor architecture design in each CPU executing parallel processing scenarios. In result, *real-time LDCs* are obtained. The FBAR application, as a sample to our information levels could be realized at the Application Software Layer in § 4.

The overall exemplars, descriptions, specifications of Fig. 2 components relative to steps:

- In Fig. 2, however, we focus on the solar panel or energy source system. Its energy provider conductors and transmitters comply with the principles used in other architectures, visible in electronics like, photodiodes which convert light to electricity based on Einstein's photoelectric effect [30, 31]. The light comes from some stellar body e.g., our Sun, and converted to an output charge into the lower level, the power subsystem components of Plan A (Fig. 1). This conveys **Routine Steps # 1 to 3**.

[2] For every action, there is an equal and opposite reaction.

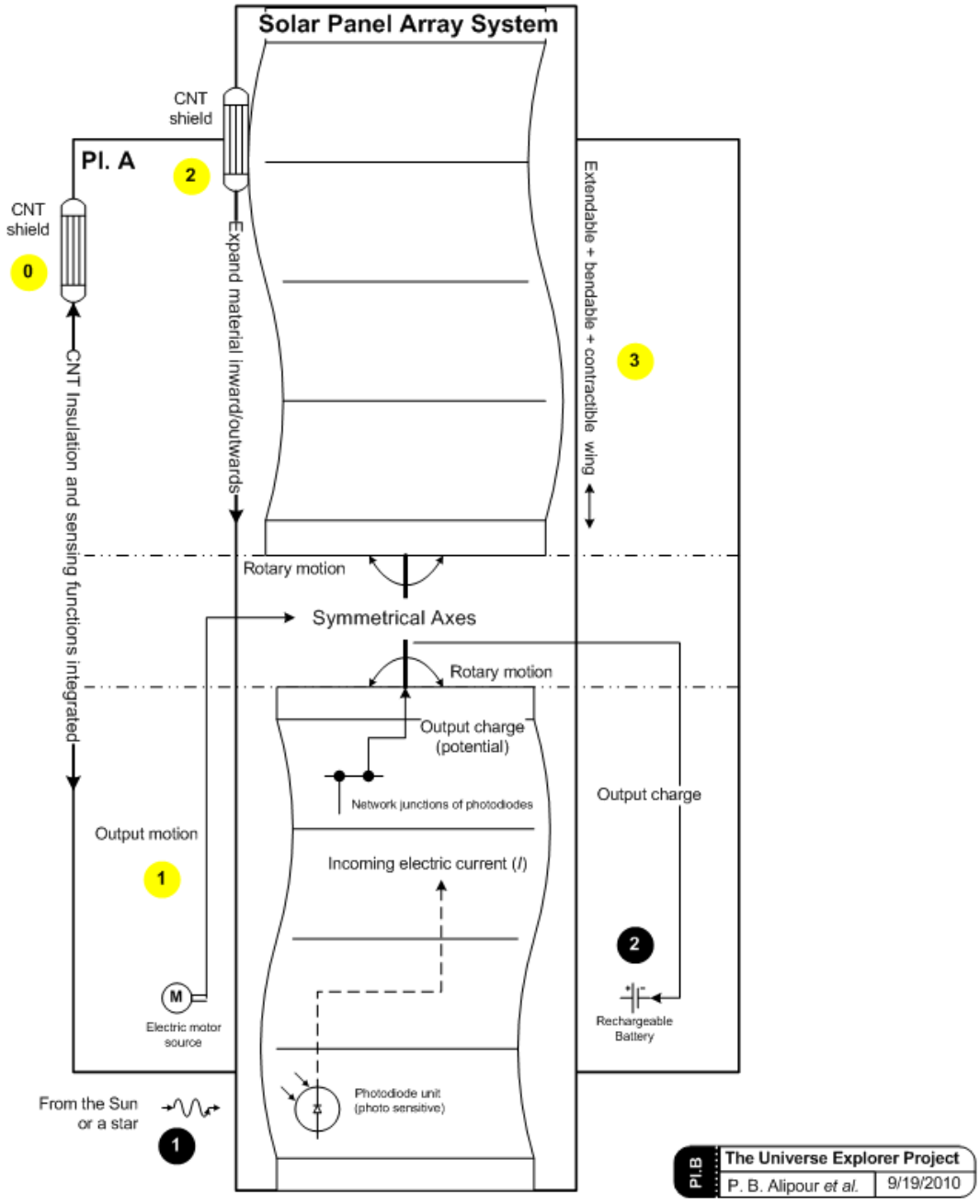

Fig. 2. The UE Satellite hardware topology (Plan B), representing the wing is comprised of solar panels (a group of solar cells) and CNT shield with relevant functions for actuation on the mechanics of the wing, signaled from the lower plan, Plan A (see Fig. 1). This plan also shows the power source for such Plan B actuations as cumulative energy converted from Sun rays to electricity or other means necessary across Plan A's system.

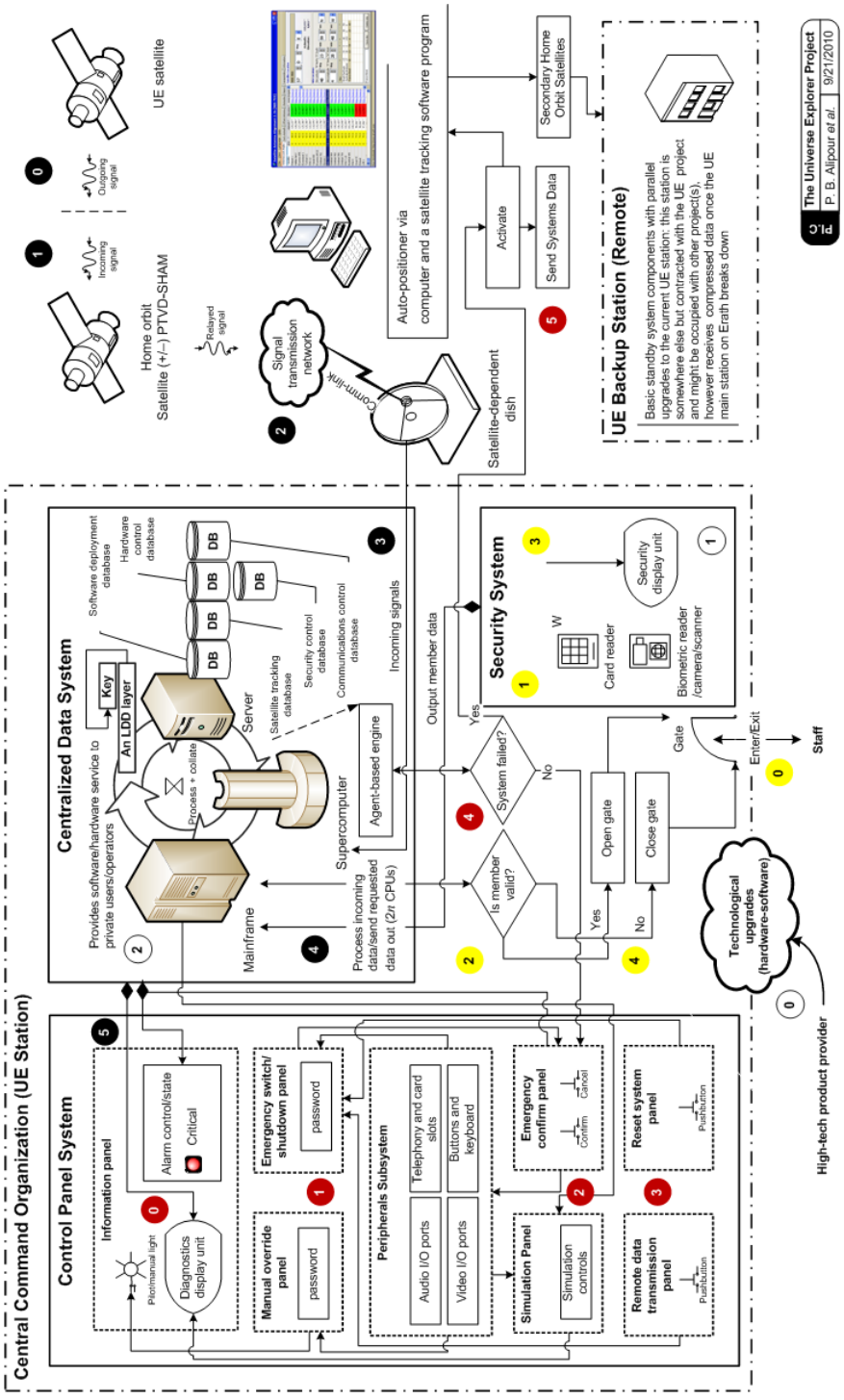

Fig. 3. The UE Station hardware and software topology on Earth, receives data from the UE satellite from space for further processing, including a control panel emergency and communications protocols, security, backup and update.

- From the lower-level, or Plan A, it is evident that one of the main power subsystem components is the electric motor source which after being supplied by electricity, converts the latter form into

rotary motion on the upper level or wing axes. This conveys **Critical Step # 1** and partly maps to the following two steps as a trigger function to their components.

- The wing could bend, contract or extend for critical situations and celestial body confrontations, like asteroids (**Critical Step # 3**). A good example is the asteroid belt [35, 36] between planets Jupiter and Mars, where it accumulates travelling bodies from other parts of space heading Earth. Jupiter's G-field traps them due to its intensity. The UE body must maneuver in case of attempting to pass through this belt when uncertain asteroid families form in this belt and thus collide. Despite of multiple unmanned spacecraft have traversed this belt without incident due to the asteroid material is so thinly distributed, still these asteroid families could cause incidents not mainly on this belt, but maybe elsewhere where the UE mission prolongs. The wing dynamics must therefore satisfy the given requirement for such situations.
- Perceivably, in **Critical Steps # 0 and 2**, we have also included the CNT shield in our design parallel to the wing's backup hardware (wing replacement). It stretches out to protect the most vulnerable parts of the wing in case of failure on structural integrity due to extremely high or low temperatures (even pressure) causing the tile to collapse beyond compensation. This is triggered by lower plan sensors and those sensors embedded within the nano-tube structure, since a CNT in its atomic junction could act like a sensitive diode or rectifier [10, 11]. It could simply conserve energy when necessary, not only for the internal system, it could also act as a physical protection cloth due to its flexibility and resistance factors. In particular, using the multi-walled version assists these physical properties in practice [12].
- For trapping space dust or interstellar dusts passing through our solar system, which UE might confront with, inclusive of thermal insulation properties,[3] in its 50-year journey, we considered the use of manmade *aerogel* (**Critical Step # 0**). The rationale to this is based on samples already in use aboard probes and spacecrafts by NASA. In particular, NASA used aerogel to trap space dust particles aboard the Stardust spacecraft. The particles vaporize on impact with solids and pass through gases, but can be trapped in aerogel. NASA also used aerogel for thermal insulation of the Mars Rover and space suits [15, 16]. The UE could collect incoming dust samples and perform further analysis by its internal sensors while on a separate issue, protecting its internal and panel integrity for its thermal insulation properties. In particular, the bus or UE body is clothed with not only different types of CNTs, and Aerogels for advanced thermal and sample collection properties. A good example as a combined material of CNT and aerogel with new emerging properties is researched as "CNT Aerogels" by Bryning *et al.* [17] (or [18]).

The overall exemplars, descriptions, specifications of Fig. 3 components relative to steps:

- In Fig. 3, the heart and brain of UE operations on Earth is illustrated. It represents software and hardware components, subsystems, storage, backup and primary-secondary data transmission protocols on display, to the UE satellite in space relative to its personnel active on ground at the control panel (**Highly-critical Steps # 0 to 4**, relative to **Regular Steps # 3 to 5**). The ambit of all controls and operations occur in a private manner. In other words, the scope of human and machine activities is limited within the boundary of the station per se. The reason to this is to avoid obvious security breach or even a system hack or infiltration. Even assuming a person tends to hack information from the wireless portion of the project (dish-satellite), against **Regular Step # 2**, the person must be equipped with a *lossless data decompression* (LDD) algorithm with an

[3] It disallows heat to escape from the UE container or entering it, whereas the shift of insulation behavior (change of position, shape, etc.) between temperature levels varies due to CNT aerogel characteristics, and thus reactive to extreme temperature data triggered from thermal sensors onboard.

FBAR key (**TT** file) to decompress incoming data as specified by Regular **Steps # 0 to 3**. Therefore, without this key and a *dynamic key* updated on a daily basis (DB dynamic index updates), external or local infiltration is impossible. This is indicated with an LDD layer accessing a static key at the Server level where real-time processing of data happens (**Regular Steps # 3 and 4**). The multi-parallel processing is performed by the supercomputer on the foreground of the centralized data system.

- To prevent latencies between parallel CPUs, we use the latest supercomputer *maintaining the speed of light communication intact*, regardless of exhibiting 1-5 microseconds latencies, based on Amdahl's law speedup factor on the optimization part is obtainable [28, 29]. These technological upgrades or componential improvements (**Future Steps # 0 to 2**, labeled with empty circles) relative to addressing bottlenecks, of course, must not intervene directly in the communications process and data transactions between DBs and UE system. Downtime, mean-time-to-failure (MTTF), mean time-between failures (MTTB), and other factors are also considered, and thus switching from a current supercomputer, mainframe or any operating server to its backup, auxiliary or array on a provisional scale, is inevitable in our design scale and integration (**Highly-critical Step # 5**). This entails a RAID system in our design implementation which provides data redundancy, fault-tolerance, data reconstruction for data retrieval purposes, error-detection and correction features, etc. In particular, we **strategize** in terms of uploading the station's compressed data to home-orbit secondary satellites, downloading all information to a second station which could be in any other geographical location occupied with other project(s) on Earth. The remote backup station has basic standby system components with parallel upgrades to the current UE station and receives compressed data in case of the UE main station on Earth breaks down. The signal bandwidth limit is not of significance, since we use the latest LDC algorithm with encoded codes for ultimate compression ratios at the station, supporting *asynchronous stream-load communication (in case of uploading to more than one secondary satellite in parallel)*, however, on-spot transmission of data to these satellites depends on the satellite tracking program for a trustable upload/download to the relative base on Earth.
- For this privatized UE station, a routine security check on staff, incoming packages or arbitrary visitors denoting **Critical Steps # 0 to 4** is considered to keep the organization assets secure enough relative to ongoing sensitive indoor operations and services.

1.2.2. An Early Core Evaluation

Choice, Rationale and Solutions

1- *Why the Usage of Two CPUs According to the Peer-reviewed Evaluation? What about having a real-time based solution in the UE design?*

Our choice and motivation: "The Latest", by definition, here denotes that, microprocessors to be built as tiny as *nanostructures*, e.g., incorporating CNTs, where one CPU could be comprised of multi-core or many-core technology [44], as well as its future generations! We could, for UE's 50-year mission say that: a *metamorphosis of von Neumann architecture* shall be recognizable amongst its subsystems. In a *scalable architecture* for its Global Analysis (§2), a manageable 2*n*-CPU hypercube *software scenarios* [48], benefiting from *software switching hypercube models* [55] for its efficient communication/transmission level, is acquirable. The hypercube in fact assists any error *codeword occurrence* by traversing its hamming

distance (Murdocca & Heuring, §9.4 [50]), and thus correcting it by the CPU. So, the many-core CPU solution is *real-time* based as well as *fault-tolerance* from quality our requirements.

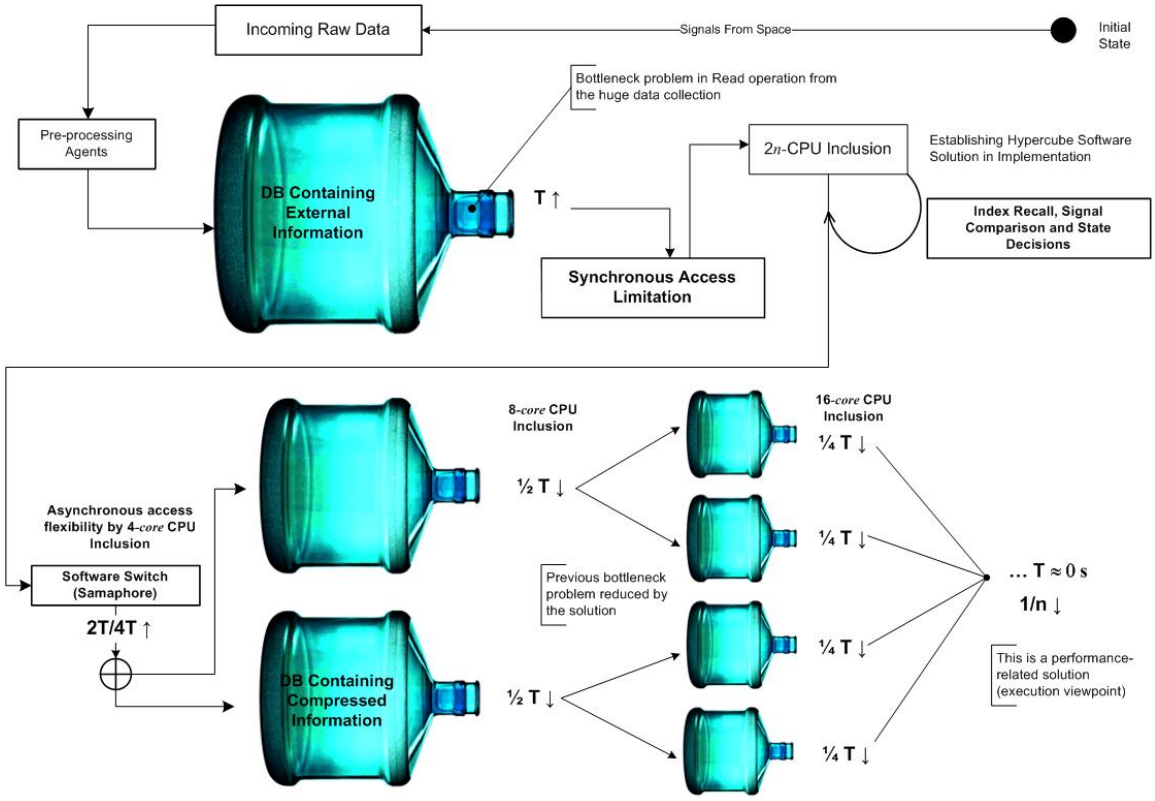

Fig. 4. A solution to our Execution View addressing bottleneck against usual architectural techniques used in the software application layer; ↓ denotes time-length reduction or decrease, and ↑ denotes time-length extension or increase

Rationale: The roles of which dedication is done either to incoming raw data parallel to onboard data. This was the initial reasoning of our choice in separating a set of dedicated 2*n*-CPU02 cores for hardware, from a dedicated set of 2*n*-CPU01 cores for incoming and outgoing signals on sight. All cores are in fact in communication, but cores from CPU02 monitor and control real-time events on cores of CPU01 as well as other spacecraft hardware components in parallel. This proves that all deadline executions are real-time, no matter how many entities are not communicating directly/*asynchronously*, since cores are in communication in parallel satisfying two time-cycles per core. The ratio of the time-cycles or r_T is an integer divided by other cores' time cycles, giving a real value approaching 0 seconds, showing a 2 time cycles divided by 2*n* real-time scenarios, or

$$r_T = \frac{2T}{2n\text{CPU} \to 2nT} = \frac{1}{n} \to 0 \text{ s} \tag{1}$$

this is an asymptotic performance solution [54] which is crucial to have in the UE system. In fact, the more cores managing data, the closer to 0 seconds executions for data receive and delivery. Therefore different UE subsystems are not bound to one CPU each, a number of CPU's or cores are used for each subsystem to read and execute more than one instruction at a time (contrary to a single core CPU) [44] relative to this

real-time bound/constraint solution. This execution behavior which satisfies parallel tasks is further covered from the same architecture view in § 5.

Inclusion: Based on the system evaluator's suggestion to have a real-time based system on-board, we have now included that all communication levels between the many-core or multi-core processors and their hardware components are real-time relative to the outgoing data to Earth, "which is not real-time". The reason is due to the increasing distance when UE reaches the far ends of its mission, latency on receiving information is inevitable due to that data must travel minutes, hours, days, [4]… light speeds to reach Earth.

Solution: We included a microchip software-hardware solution onboard the satellite to perform real-time between UE and Earth when far from home in space. This is well-explicated in §2.1, instead of waiting for hours, days, … for the current information onboard to reach the base on Earth, bending time in terms of superposition of the data carriers, or hot electrons [47] designed in the memory (with a tuner on the send/receive frequency) The bottleneck problem is further resolved in terms of sub-tasking in cycles on a *software level* (see paragraph on motivation) as explicated by the above equation using 2*n*-CPU cycles. This is hereby illustrated in Fig. 4, above:

This bottleneck solution in Fig. 4, on shared memory and/or accumulated data, is included in our Execution View in terms of *compressed space* and *static space* (memory), §5, specifically portraying the importance of how two main processors could work together in parallel dealing with specific data for a read/write operation, asynchronously. One main processor deals with priority data as raw values only, and the other, deals with assignments to the given tasks on prioritized signals onboard the satellite. Both operate asynchronously with a *semaphore* without affecting the performance, since no queue carrying state signals for the hardware controller is delayed or overflowed against the queue on incoming/outgoing raw data between Earth and Space. If so, the priority is already privileged by the 'software switch' in the hypercube of processors (first paragraph above), which means no interference of real-time critical tasks against data compression and relay tasks explicated in §5 due to having alternative routed and communication paths (or Fig. ExV4, which also benefits Eq. 1). One last not would also be that tasks allocated for the one main processor does not have to wait for the tasks allocated for its parallel (other main processor). In other words, the group of certain tasks handle operations related to hardware managements, whilst the remaining focus on just passing the information no matter how minimal or maximal in deliverables to/from Earth during executions.

Therefore, there will be 'no design transformation' of the current core architecture based on this evaluation and merely as a refinement of the architecture on the views. (Later included in § 2.1.7.)

2- *Will the database get full during transmissions especially when the UE learning-based system grows in space?*

For the learning-based system, the new nodes that are created and added to its agent-based intelligent network (as software nodes), some of the older nodes become abstracted or replaced with shorter executables, thus keeping only vital instruction intact, and the rest, relayed to Earth or erased. This, of course, entails the processing solution brought up in Q.1. It learns not only from errors, for the space environment and previous experiences (scenarios like e.g., Bob in § 2.1), next time, similar conditions will not apply to the UE software-hardware failures in space. This has been mainly discussed, and exemplified,

[4] It relatively takes (½ day – 8.3 minutes), for data to reach Earth, if UE is on the other side of our solar system.

with our pseudocodes in §6.1.2 from a Code Architecture viewpoint, which conveys to adaptable UE scenarios in Space, and thus, its efficiency in communications, both *internal* (onboard circuitry) and *external* (from/to Earth). On the external communication level between Space and Earth, there certain space-time physics barrier points, or limitations of communication, which are inevitable during the UE mission in space. This particular issue is addressed in §2.1 of our Global Analysis worst-case and normal-case scenario factors (or see, Eq. 3 with its relative discussions).

Therefore, there will be no design transformation of the current core system of the UE architecture based on this evaluation. However, a refinement of the current architecture pointing out how strong the design has integrated such solutions inclusive of any potential flaw (e.g. on the time-delay or efficiency part) later included in § 2.1.7. We further emphasized that we have already considered in our initial choice of the design on views through our strategies satisfying quality requirements during analysis.

Further relative evaluation is addressed in §6.1.3 of the current report.

1.2.3. UML Use Case Diagrams for UE Hardware-Software Specification:

A Preamble to Global Analysis Scenario Factors

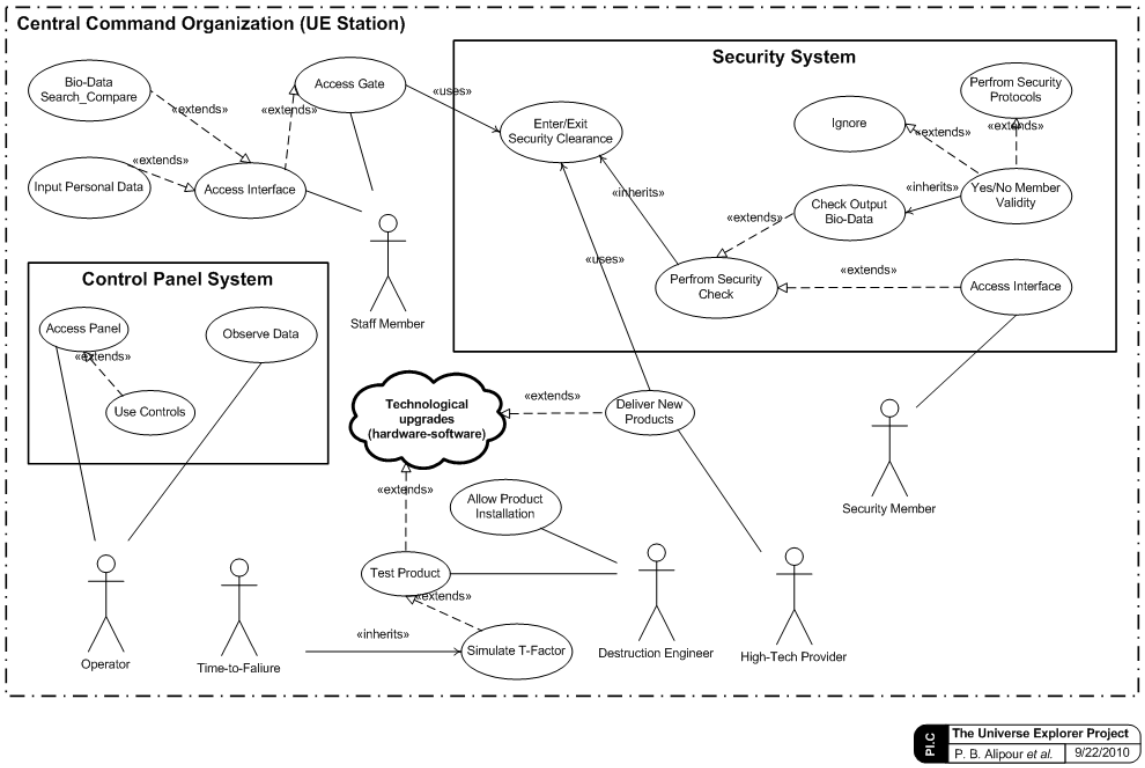

Fig. 5. This is an extension to Fig. 3. This figure represents a UML Use Case diagram in terms of legitimate actors of the system. All of these actors, apart from the inherited time task called Time-to-Failure, indicate certain actions in a cyclical manner from bottom tasks to top-level task in the system. The "Time-to-Failure" actor is indeed unique as

performing an inheritable subtask activity on hardware-software components. It initiates when a component is destroyed, thus making the Destruction Engineer (the co-actor), aware of the functioning limit to the application, before being installed onto the UE's centralized data system and elsewhere (Fig. 3).

Fig. 5. The Control Panel + Space-Earth-Space Use Case (*narrative style*):

Name: Access Control Panel
Identifier: UC 19
Description: Let operator access the Control Panel.
Pre-conditions:

- The Operator is logged into the system.
- The Operator has gone into the initial checks to verify that he/she is the person eligible to access the control panel (redirected from the Security Use Case)

Post-conditions:

- The operator on Earth will be enabled in the station as a UE expert to use controls parallel/relative to data observation from UE in space.

Basic Course of Action:

1. An operator wants to access controls.
2. The UE in space delivers data to the UE station on Earth.
3. The operator visually inspects incoming data whilst being processed.
4. The operator determines that the data is showing a new UE system behavior in space.
5. The operator uses relevant controls to access system core for addressing this issue.
6. The operator sends relevant data via system core to UE system for carrying out new instructions. (an extension is given in Fig. 5)
7. The use case ends.

An extension to Fig. 5 illustrating other use cases, show other legitimate actors of the system in Space and Earth in-communication. All of these actors are unique and have strong technological dependencies to one-another. The "Time" actor is indeed unique as performing an inheritable subtask activity on hardware-software components for a signal send-receive transmission. It initiates when a component is active for deliverables parallel to receptions between Space and Earth, thus making the operator at the station, aware of the functioning limit to any reception, otherwise uplink conditions (see, §2.1 scenario) to the UE system are proceeded. A concealed co-actor is evidently a compressor which initiates a lossless data compression (LDC) considering data space and latency issues for processing it, which are tackled with according to technological upgrades as a solution to the light speed transmission limit (see next section).

2. Global Analysis

The key characteristics that must be met by the UE engineering system can be classified into *human computer interaction (HCI) factious scenarios* at the Earth and space levels, *product factors*, *technological factors*, and *organizational factors* entailing *security factors* during/after development. The "factious scenario", here, we mean disobedient or rebellious situation occurrences, either on purpose or otherwise e.g., some security breach issue, or some human error at the control panel, or some erratic program behavior from space or the system, mechanical failure, etc., which are all against the healthy system's status report: ROM-based real-time diagnostics, checking UE hardware and software components. Such scenarios are significant for our global analysis as the UE's system scenario factors indeed. We commence with analyzing a scenario against a character called Bob working at the UE station on Earth experiencing problems from space where his expertise will come handy as a trained employee at the center. In fact, we

execute a scenario as an algorithm with actors involved for a Yes/No already-stipulated and answered actions from one state of action to another (its subsequent). It is thus, the plan for its engineering predecessors, UE design pioneers and post-conditions to assess how the situation was handled under such extreme conditions with respect to time and global action report(s). In the following section, one could realize that many factors defined in the ISO/EIC 9126-1 software quality model with six main quality characteristics: *functionality reliability*, *usability*, *efficiency*, *maintainability*, *portability*, are identifiable throughout the course of Bob-UE system-Bob actions and reactions. For instance, from the usability perspective, overhead information [37] is managed by the control panel user or "Bob" correcting errors aboard the spacecraft through a simulation pad identifiable in Fig. 3.

2.1. System Scenario Factors, Limitations and Solution

Bob, the UE central command operator has normal sight and no physical or perceptual impairments onboard the satellite when he studies his monitors and diagnostics screens (Fig. 3). His shift started at 11pm and it is now 5am in the morning. So far, the command center has been operating within normal parameters and the current alarm state is therefore green.

Assumptions: Assuming the ongoing events in the satellite are delivered as delayed data at some point of time close to 5 minutes, since the satellite transmits data at the speed of light[5] and now is approaching Mercury, thus it takes more than 4 minutes for its real-time status report data to reach the station. Luckily Mercury is at its closest point to Earth.[6] The UE has now, successfully completed its mission over Mercury approaching an orbit slingshot via Venus towards the remaining planets before exiting our solar system (similar to Cassini probe mission [27]), which could take years after its first exit from the Sun's gravitational field to the ends of our solar system.

1. Bob notices the internal temperature of the UE bus-chamber has risen very rapidly.
2. He also notices for some odd gravitational shift between nearby planets Mercury and Venus due to the after-effect of a solar burst/flare from the Sun (hard to predict [24]), UE in consequence has altered its course wrongly-directly toward the Sun!
3. The diagnostics show that the intense electromagnetic radiation caused the unit to miscompute the actual course with the wrong one as correct data!
4. The component controller subsystem onboard has already executed necessary temperature reduction protocols via the depressurizing system with its fluid CNT coolers from both inside and the shield (refer to Fig. 1's yellow signal flow).
5. But the system fails to automatically navigate the UE due to some hardware-software failure in commanding two of the thrusters to avoid getting too close to the Sun.
6. Bob realizes he must immediately change the satellite course manually and target the right direction by sending the overriding coordinates to the UE system to correct this
7. Luckily, the simulation pad on the control panel is connected to the mainframe server and has the predetermined coordinates relative to planets, celestial bodies, objects, etc. in space, so Bob's job won't be errored in trajectory calculations, and thus UE course correction telemetry.[7]
8. So, he runs the simulation program and inputs the coordinates accordingly, once he estimates with his own manual calculations for comparisons, if a genius not to waste further time!

---

[5] The *Speed of light* or *Celeritas* as a so-called electromagnetic constant c or light speed in Physics, in terms of $v_{light} = c \approx 300{,}000$ Km/s.

[6] 77 million kilometers / speed of light ≈ 4.28 minutes; this is a normal-case scenario, unavoidable to have in the worst-case experienced by Bob.

[7] Telemetry is vital in the development phase of missiles, satellites and aircraft because the system might be destroyed after/during the test. Engineers need critical system parameters to analyze (and improve) the performance of the system. Without telemetry, these data would often be unavailable. (See also [32].)

9. He goes to the Alarm Control Panel on the far right of the main control panel and presses '+' twice (as it is starting off in green state)
10. This allows other experts to be alerted near or in the vicinity of the station and thus help Bob, just in case: the station has no means of global network communication due to the sensitivity of the project and all personnel for such situations must be checked-in and operate on site. However, the system is open to secondary security protocols in case of disaster-related issues to switch from this station to another geographically-located remote station, in aim of continuing operations on its behalf.
11. The Emergency Confirm button glows red
12. He moves across to the 'Manual Override' panel on the far-left of the main station control panel.
13. He selects 'Manual Pilot' from the pull-down on the 'Manual Override' panel.
14. He types the new value '1503' using the keypad (*this is a standardized UE password scenario, 15 for 'O for override' from the order of English alphabet letters, 03 staging 'error type 3 or critical', hence the combination sys: Override Critical Stage now!*)
15. All thrusters are displayed for manual control
16. He navigates EU's on the Simulation Panel by an auxiliary component controller (backup) system while CPU 02 operates on most functions except the thrusters due to actuation signal conversion failure.
17. He uplinks the entire vital trajectory coordinates to the satellite which could take more than 4.3 minutes to reach the satellite instructions CPU sytem.*
18. The computer on Earth calculates that the breakdown of the internal system is imminent after it gets really close to the sun passing Mercury's orbit, as UE enters sun's vicinity with 2000 ^{o}C - 6000 ^{o}C range, in < 15 days. Mercury-Sun distance/ climbing speed of 25km/s (maximum threshold for thrusters to fall into the right course or orbit is 40 km/s), ergo, (58 million km) / (40^{+} (km / s)) < 16.78 days.
19. He notices that the navigation number on the UE Navigation Target panel has not changed.
20. He realizes he forgot to press the SET button on the Manual Override panel.
21. He presses the SET button for executing the coordinates.
22. Another 4.3 minutes has elapsed.
23. An automatic audio warning sounds "20 minutes to UE trap zone" and it further displays that the satellite is now accelerating jumping from 25 km/s to 38^{+} due to Sun's gravitational field influence on UE via planet Mercury
24. All alerting systems suggest that the UE must make its move now, since this is the last chance to perform below threshold points of an *escape velocity* V_e= 67.7 km/s, to sufficiently escape Mercury's gravity relative to Sun's gravity by falling back to Mercury's course, from its current position, otherwise, since there are no rockets or strong propulsion systems to act on behalf of the thrusters, UE is absolutely doomed for the remaining days of its journey!
    Note: We assume the thrusters support a maximum of 40 km/s with a good force *F* push for an orbit slingshot (gravity assist maneuver) outwards to Venus's orbit without damaging the neighboring equipments. On the other hand, if the UE enters layers close to the coolest layer above Sun's photosphere > 500 km with temperatures ≤ 4100 Kelvins ≤ 3826.85 ^{o}C, [25, 26], and if not change its course to the right direction way before then, it will get trapped in Sun's gravity for good. This is indeed UE's final demise beyond its capable CNT aerogel and other shield materials!
25. Bob has now done whatever he could do, and hopes all the coordinates for the thrusters including backup balancers function for the right change of course.
26. He'll find out after approximately 4.5 minutes what is UE's current status, and still the red light is ON.
27. Bob receives incoming data from Sun gathered by UE after 4.5 minutes, including UE's course change, telemetry updates from Bob, and UE unit responses onboard the satellite.
28. It shows that UE successfully escaped by launching the right course of action, not only on its current thrusters, a slingshot on Mercury's orbit to escape the trap with temperature control for all its systems through Venus
29. The audio system announces "UE navigation override protocol successful"
30. Incoming diagnostics for the next parts of UE's mission from the satellite is now in position.

31. Valuable data is now stored in the compressed DB for further analysis on the planets, including Sun and the *initial flare* causing all of these problems, handled by Bob as a competent employee working at the station despite of his natural human errors.

*** Limitations to be rectified during the UE's 50-year mission:** *Bending space and time* for the incoming and outgoing signals in the coming years could be achieved by e.g., particle accelerators, in particular, *tachyon emission travelling faster than the speed of light* [19-21], but with safe energy scales between Earth and the UE unit in terms of the given spectra range supported by it onboard sensory components (See, Fig. 1).

A pre-emptive solution: P. B. Alipour, in his *pre-print papers*, back in 2007 [22, 23], designed a *Satellite Comlink in form of a super-helical CNT microchip* or PTVD-SHAM (Figs. 1.2 and 2.1, [23]), creating a magnetic field using the 2D electron gas and Faraday effect to collect data using parallel memory cells exhibiting speeds of 2*n***c** (greater than light speed or factors of **c**), conjecturing possible parallel time-varying properties, and thereby transmit it back to Earth real-time (or much sooner than the expected latency). In Fig. 3, this has been depicted on the home orbit satellite as a technological upgrade product. In the UE data transmission scenarios, it delivers compressed imagery data after receiving from space, to be reconstructed on-chip (CCD) via the supercomputer at the station. The technique focuses on the way electron or charge representing compressed data is bailed-in and out into two places at the same time (superposing). Therefore, data could by a contractible vs. expandable distance between satellite and receiver/sender on Earth, using a tuning device mounted on the PTVD-SHAM microchip, such that these carriers (hot electrons *e*) traverse memory cells on both systems UE in space, and home orbit satellite simultaneously (compare with Fig. 4.2 [23]). This theoretically gives the expected speeds of computation quite higher than the speed of light. For *the time-varying* process causing this to happen, assuming that we have this microchip onboard the UE satellite, Bob's transmission scenario for his uplink procedure would be quite reliable in reducing the estimated 4.3 minutes delay, in commanding the Satellite, down to just a few seconds when technology progresses in virtue of signal application upgrades.

——— *"Assume in the laws of quantum physics that, any assigned value as an input is pondered to bring about the notion of 'uncertainty'. On the contrary, conjecturing the notion of certain reserved locations for data by pre-defined time in entanglement (predetermined on anticipated time) for uncertain values, is itself upon any individual which tends to read/write data and time, understood as a 'certainty' "!* [23] ———

So, this is already designed in theory [22], whereby a well-defined funding by the UE space organization on this microchip, is deemed to be processed and finalized for the manufacturing /fabrication phase of the time delay-saving application. In fact, the PTVD chip stores $\boldsymbol{T_{delays}}$ and at the *application layer*, gains a conversive function to real-time via its *algorithm* = $t_{\text{delay1}} + t_{\text{delay2}} + t_{\text{delay3}} + \ldots + t_{\text{delay } n}$ with respect to $data_{\text{in}}$ (incoming from the space environment), whilst intersected with (or multiplied by) $data_{\text{out}}$ (outgoing data to Earth), with respect to current time (Earth's time standard experienced at the control panel), or

$$PTVD(T_{delay}) = T_{real} = \frac{t_{delay1} + t_{delay2} + \cdots + t_n}{data_{in}} \times \frac{data_{out}}{t_{current}} = 1 \qquad (3)$$

This occurs, if and only if, hot *e*-carriers representing data are superposed in location at light speed. This relationship is elicited from [23] and thus gives a time constant K_t equal to 1 denoting real-time data transmission (§ 2.2 of [23]).

2.1.1. Analyze Product Factors:

The Building Blocks for the Conceptual View

We have listed important product factors for the UE system (Figs. 1 and 2) in Table 1.

UE Product Factors (Part A: Space)

Product Factor	Flexibility and Changeability	Impact
P1: UE satellite features		
P1.1 Application policy rules		
Support software upgrade, settings and new application policy rules.	New settings, instructions and rules are added when necessary.	There is a large impact on all components.
P1.2 Transportable main unit		
The UE bus or whole body is transportable based on navigation subsystem and/or interactive physical forces in space. (The subsystem is supported by life-time battery or power source.)	The body must adjust dynamically to transport in space while not affecting software-hardware performance.	This affects signal acquisition, and processing relative to change of location in space.
P1.3 Critical detection of I/O devices		
The UE satellite must communicate with Earth (UE station) while onboard intercommunications between components monitored, processed and managed.	New forms of data are received and updated.	This affects signal acquisition when critical situation from space environment or application upgrades from Earth, and there is a large impact on all components if environmental threats as new data are experienced.
P1.4 Dynamic detection of I/O devices		
The software must detect the presence of I/O devices (CPU 02).	The CPU monitoring or control system must handle dynamically changing features based on sensors or connected internal-external devices.	This affects all targeted components relative to the space environment flux.
P1.5 Scalable architecture		
The UE engineering system must be scalable to support a vital range of software while backup hardware remains intact.	This is fairly stable via the partitioned database onboard for software upgrades; this is unstable when hardware failure where alternative backup systems or replacements restore stability until subsequent failures (if any).	This impact affects primarily the conceptual view.
P.2 User interface		

(dismissed)		
P.3 Performance		
P3.1 Real-time performance		
There must be real-time processing of raw data aboard the satellite and display data on Earth. Each data type has a maximum delay specified in terms of light speed factors externally, and internally, close to 0s scenarios.	The maximum delays are negotiable in terms of signal compensators relative to the LDC application that provide efficiently compressed data to the compensator; special microprocessor and memory designs customized on board and upgradable Earth-orbiting satellites on hardware technology, is contemplatively imperative in design.	There is a large impact on signal acquisition and processing components in terms of data reception and relay signals.
P3.2 Processing deadlines		
Several processing priority levels are needed to support multiple processing deadlines.	Deadlines and priority levels could change based on changes to the requirements or platform before UE launch on Earth. These levels could also change based on unique instructions upgrades in form of Assembly or machine codes uplinked from Earth to space.	There is a large impact on all components if process partitioning changes.
P3.3 Processing parallelisms		
Several parallel processing priority levels are needed to support multiple parallel processing executions.	Executions and parallel priority levels could change based on changes to the requirements or platform before UE launch on Earth. These levels could also change based on unique instructions upgrades in form of Assembly or machine codes uplinked from Earth to space.	There is a large impact on all components if parallel processing relative to partitioning change in operation synchronicities between hardware components.
P.4 Dependability		
P4.1 High availability		
The system must display correct information on Earth, 24 hours a day, and 7 days a week.	Requirements are stable if UE explores a nearby planet. The farther the satellite explores in space, the less stable due to information light speed with respect to distance barrier. Thus knowing what is happening real-time at that point comes too late afterwards. (Bending time relative to space with special	There is a large impact on all components concerning their real-time status when UE is far from Earth. There is a moderate impact on all components when nearby.

	microchip design on Earth as compensators could make this compulsory delay near to real-time data receptions)	
P4.2 Withstand radiation, spatial disturbances, forces, objects, etc.		
The UE subsystem must be impervious to imminent interruptions such as extreme pressure, temperature, radiation, gravitation, etc. in space and avoid physical impacts through intelligent means on its components.	There is no flexibility and this requirement could be satisfied with currently available adaptive components such as CNTs.	This affects high-tech hardware selection.
P4.4 Software quality		
The UE deployment projects must conform to regulatory efficient requirements for software quality for efficient ways of communication without any data loss.	There are different ways this could be fulfilled such as double-efficient LDCs giving fixed time and data compaction ratios, relative to *equipartitioned data* (clusters), etc. via relevant applications installed onboard.	This affects nearly all design decisions on a software level.
P4.5 Advanced-level implementation		
The implementation of projects on Earth before lunch must be as sufficient and efficient as possible with latest technologies including ongoing researched materials.	There are advanced scientific ways this could be fulfilled.	This affects all views.
P.7 Product cost		
P7.1 Hardware costs		
Hardware costs for projects must support latest-based research at both laboratorial and current-based hardware pricing. Cost of almost all components is flexible.	There is in most cases moderate flexibility below the spacecraft budgetary limit.	There is a large impact on all components.
P7.2 Project development costs		
The cost of verifying, validating, and licensing projects must not prohibit vital product pricing.	It is prohibitively expensive to verify, validate, and license each project independently.	Strategies are needed for reducing these costs.

Table 1. Product factors influencing the UE system.

In continue, we have listed important product factors for the UE central command systems (CCS) on Earth (Fig. 3) i.e., control panel, centralized data and security systems, in Table 2.

UE Product Factors (Part B: Earth)

Product Factor	Flexibility and Changeability	Impact
P1: UE CCS features		
P1.1 Application policy rules		
Support software and hardware upgrade, settings and new application policy rules.	New settings, instructions and rules are added when necessary.	There is a large impact on all components.
P1.2 Restorable main units		
The UE centralized data and control panel systems must be replaceable at any time in case of hardware failures without affecting current operations.	Apart from components being repaired based on MTBF factor, there is no flexibility and damaged components must be replaced.	There is a large impact on almost all components if there is no backup system considered.
P1.3 Critical detection of I/O devices		
The CCS systems must communicate with space (UE satellite) while intercommunications between components monitored, processed and managed.	New forms of data are received and DB Server is updated.	This affects acquisition, storage and processing of CCS data center; this also affects control panel and user-based/artificial-intelligence-based decisions.
P1.4 Dynamic detection of I/O devices		
The software must detect the presence of I/O devices (CPU 02).	The CPU monitoring or control system must handle dynamically changing features based on sensors or connected internal-external devices.	This affects all targeted components relative to the space environment flux.
P1.5 Scalable architecture		
The CCS engineering system must be scalable to support a vital range of software while backup hardware remains intact including hardware-software technological upgrades to the center.	This is fairly stable for software/hardware upgrades; this is unstable when hardware failure where alternative backup systems or replacements are not invested or managed.	This impact affects primarily the conceptual view.
P.3 User Interface		
P3.1 Accessible UE space engineering, information and navigation tools		
The scientists, engineers and operators at the CCS systems must be provided with tools to communicate with each other as well as the UE satellite with short and long term verification and control performances on internal-external data.	This is fairly stable depending on the UE condition in space, incoming data and security.	There is a localized impact.
P3.2 Processing deadlines		
Several processing priority levels are needed to support multiple processing deadlines.	Deadlines and priority levels could change based on changes to the requirements or platform	There is a large impact on all components if process partitioning changes.

	before/after UE launch. These levels could also change based on unique instructions upgrades in form of decompressed data at the station.	
P.3 Performance		
P3.1 Real-time performance		
There must be real-time processing of raw data at the station. Each data type has a maximum delay specified in terms of light speed factors when reaching the station.	The maximum delays are negotiable in terms of signal compensators or special microprocessor and memory designs customized on board and upgradable Earth orbiting satellites on hardware technology.	There is a large impact on signal acquisition and processing components in terms of data reception and relay signals.
P3.2 Processing deadlines		
Several processing priority levels are needed to support multiple processing deadlines.	Deadlines and priority levels could change based on changes to the requirements or platform during development.	There is a large impact on all components if process partitioning changes.
P3.3 Processing parallelisms		
Several parallel processing priority levels are needed to support multiple parallel processing executions.	Executions and parallel priority levels could change based on changes to the requirements or platform during development. These levels could also change based on unique instructions by software upgrades.	There is a large impact on all components if parallel processing relative to partitioning change in operation synchronicities between hardware components.
P.4 Dependability		
P4.1 High availability		
The system must display correct information on Earth, 24 hours a day, and 7 days a week.	Requirements are stable if UE explores nearby planet. The farther the satellite explores in space, the less stable due to information light speed with respect to distance barrier. (Bending time relative to space with special microchip design on Earth as compensators could make this compulsory delay near to real-time data receptions)	There is a large impact on all components concerning their real-time status when UE is far from Earth. There is a moderate impact on all components when nearby.
P4.2 Withstand disaster-related and/or security breach issues		
The CCS systems must be impervious to natural, environmental and human-disaster-related interruptions such as sabotage, weapons of	There is no flexibility and this requirement could be satisfied with currently available technologies on material physics and architecture.	This affects civil engineering, as well as hardware-software engineering material selection and design on both backup and current systems targeted

mass destruction, earthquakes, hurricanes, etc. on Earth.		components.
P4.4 Software quality		
The CSS relative to UE deployment projects must conform to regulatory efficient requirements for software quality for efficient ways of communication without any data loss.	There are different ways this could be fulfilled such as double-efficient LDCs giving fixed time and data compaction ratios, relative to *equipartitioned data* (clusters), etc. via relevant applications installed at the station.	This affects nearly all design decisions on a software level.
P4.5 Advanced-level implementation		
The implementation of projects for the satiation on Earth must be as sufficient and efficient as possible with latest technologies.	There are managerial engineering ways this could be fulfilled.	This affects all views.
P.7 Product cost		
P7.1 Hardware costs		
Hardware costs for projects must support latest-based research at both laboratorial and current-based hardware pricing. Cost of some components at the station like the mainframe or supercomputer is flexible.	There is in most cases moderate flexibility below the CCS systems budgetary limit.	There is a large impact on all components.
P7.2 Project development costs		
The cost of verifying, validating, and licensing projects must not prohibit vital product pricing.	It is prohibitively expensive to verify, validate, and license each project independently.	Strategies are needed for reducing these costs.

CCS = UE centralized command systems which comprises of centralized data control panel and security systems.

Table 2. Product factors influencing UE CCS systems.

The features factor provides the product requirements visible to the user at the station parallel to the satellite before and after launch. The product supports thousands of customized engineering, policy rules and cutting-edge technologies in terms of hardware and software, while all devices onboard the UE are controlled, monitored and managed by are mostly software applications.

- The duration for data transmission varies as pointed out in our scenario factors, § 2.1, for the external data transmission factors from space to Earth. The reason, as determined and specified, is that during the 50 year mission, such as the speed of light barrier is always in place relative to the far distance travelled by the UE.
- No matter how we design the transmitters, the acquisition devices must remain in a standby mode in terms of minutes, hours or even days to receive information from space to Earth.
    - To this account, light-weighted CNT materials were strong candidates to even manufacture the main parts of our sensors, CPU, capacitors, etc., making the job of our

system *entities* much easier than ever for data transmission and thus communication paths, down to the execution level.

- Thus, the 2*n*-core CPU solution is plausible due to the compactness of the hardware product incorporating CNTs, and thereby for its software and instruction set-handling, we **strategized** in terms of partitioning any bottlenecks emerging from the BDB system for system processors, which is already alleviated with the fixed LDC solution (recall §1.2.2).
- This results in fixed execution points between static access of data points and hardware components. In particular, based on the findings made by Alipour on the fixed LDC [6], for an incoming flow of 61kB data (maximum allocation), gives a maximum delay of 0.3 sec. for 87.5% compression (x86 architecture) with a static information ≈ 8MB satisfies 256×256= 65,536 unique combinations.[8] This is quite suitable for RGB true-color (based on high-color {R}, {G}, {B} discrete 1x2 setting, now intersecting as 1x3 setting) combinations, once image content sent in *red*, *blue*, and *green* to earth. Once received from this end, mixing them to produce high resolution images is an easy task.
- The *n*-core CPU solution also ensues solutions on direct error correction issues rather than "*detection then correction*" issues by software evolutionary factors, § 2.1.4, such as the ABL algorithmic sample, analyzed and explicated in our Code View architecture § 6.

- Furthermore, the idea to incorporate PTVD-SHAM hardware component in our system, is hypothetically-extremely useful to address the light-speed problem causing latency issues, transforming UE into a real-time system, for our communication protocols with the satellite. The following table, further expands on the specified duration limits of our system, both in Space and Earth raised in its product factors.

Processing deadline	Maximum Delay	Delay Reduction by 2*n*-Core Inclusion with relative Adaptive Software Algorithms (Semaphores)
In Space		
Main signal state processing delay	0.8 s	800 msec.→≈0 s
Compressed state processing delay	0.4 s	400 msec.→≈0 s
Local waveform processing delay with compression	0.2 ~ 0.3 s	200~300 msec.→≈0 s
Relay pulse delay to Earth	1.1 s to ∞ s	∞→≈0 s
On Earth		
Local processing delay on Earth apart from simulation processing *	0.4 s	400 msec.→≈0 s
Drift in blip/beep stream	50 msec.	50 msec. →≈0 s
Alarm annunciation delay	500 msec.	500 msec.→≈0 s
Relay pulse delay to Space	1 s to ∞ s	∞→≈0 s

Table 3. UE Performance expectation in space relative to Earth.

* *Simulation processing* depends on the input variables and the newly-arrived information from UE, which could vary up to a few minutes relative to real-time human-based decisions. It also heavily depends on the size of the *decompressed data* after receiving compressed data concerning the UE status in space for a solution.

For example, the *relay pulse delay*, depends on the maximum permitted delay with a standard CPU or worst-case scenarios happening aboard the satellite. Hence, the 1.1 sec. delay is logical, assuming the two parallel main processors behave concurrently, the one dedicated to data compression is disregarded, thus

[8] Each integer value represents one pixel in the combinations' set, and compared to textual type LDC giving 50% compression ratio, this would be indeed significant (for the latter, ≈ 32 MB is required according to [6]).

giving 0.8 + 0.3 = 1.1 seconds. We have therefore applied the same logic to the systems here at the station to communicate with UE in space.

The product factors related to performance refer to the real-time processing requirements of the system. The compression technology adhere is a solution to packetized local information between process steps as *filters* to maintain this performance at its highest level possible (recall Fig. 1, Step # 3 relative to description iii). We now introduce the technological factors corroborating with the above table.

2.1.2. Analyze Technological Factors:

The Building Blocks for the Conceptual View

UE Technological Factors

Technological Factor	Flexibility and Changeability	Impact
T1: General-purpose Hardware		
T1.1 Processor type		
The system has a signal processor, two main parallel 2*n*-core processors, one dedicated to BDB transactions, and the other for voting and monitoring hardware components. All of these processors are in a collection of the 2*n*-core technology.	For the main processor, the CPU type is likely to change during the 10 year development cycle (before launch.)	There is no impact unless the custom *real-time operating system* (UE RTOS) changes
T1.2 Processor number		
2*n*-Core processors, two main processors onboard the UE for specific applications	The number of cores may increase before launch	There is a need to support distributed computing.
T1.3 Memory and database		
A base fixed 32MB memory for data compression. For sensory devices and many-core CPU technology, we proportionally require 2*n*-32MB as well as their relative code. For relay, a PTVD-SHAM concentrates stored information with low speed (like flash memory) to generate a barrier-breaking speed on the signal (accelerating it). For static access and change of controllers, we considered the usage of EPROMs.	There is no flexibility once UE is in space	This places restrictions on implementation.
T1.4 Diskless system as backup		

All sensory devices, super-capacitors could store information in case of fresh or checkpoint backups/restoration, in form of standard data packets when necessary.	There is no flexibility once UE is in space	This, to some degree mitigates restrictions on T1.3
T.2: Domain-specific Hardware		
T2.1 Input/output system		
There are many types of input and output devices onboard the UE.	On the software side for new types and upgradability to manage this domain is flexible (see T3.1). On hardware, no flexibility.	There is a large impact on acquisition components
T2.2 Processing deadlines		
Several processing priority levels are needed to support multiple processing deadlines.	Deadlines and priority levels could change based on changes to the requirements or platform before UE launch on Earth. These levels could also change based on unique instructions upgrades in form of Assembly or machine codes uplinked from Earth to space.	There is a large impact on all components if process partitioning changes.
T3: Software technology		
T3.1 Distribution		
The UE RTOS operating system must support software distribution from the server or services provided at the station on Earth. The local distribution on the client/server part at the station on Earth must provide support by its operating system.	If the operating system doesn't support this, the programmed algorithm (e.g. agent-based learner or ABL) must provide a minimum service for evolving the operating-system. The operating system at the station on the other hand, must provide these local services.	Additional components during uplinks (upgrades/new packages) to UE in space, as well as local operations at the station on Earth are needed.
T3.2 Reentrant code		
The custom operating system should provide full-support for multiprocessing.	If the operating system doesn't support this, portions of the software must be written as reentrant code.	It is difficult to write correct reentrant code.
T3.3 Code portability		
The product must be ported in form of uplinks from Earth for the custom operating system, which could force a change in software platform.	A change in hardware is not possible. Adaptive physics scenarios from space could emerge, or better data management issues (like a newly crafted compatible	On the main processor, there is large impact on everything using UE RTOS and inter-process communication (IPC) services. The relay and acquisition components

	compression technology) which forces a change in the software platform.	relatively allude to the changes made in scalability of compression that entails updated code combinations.

Table 4. Technological factors influencing the UE system.

2.1.3. Analyze Organizational Factors:

The Building Blocks for the Conceptual View

UE Organizational Factors

Organizational Factor	Flexibility and Changeability	Impact
O1: Management		
O1.1 Support for reuse		
There is a preference for in-built or self-embedded use software but no support for reusability.	Reuse of domain-specific software components built-in the UE system will be considered.	There is moderate impact on meeting the schedule for code updates.
O.2: Staff skills		
O2.1 Range of training		
The UE project requires expertise in process engineering, electrical engineering, astrophysics, mechanical engineering and software engineering.	Other staff could be transferred to the UE project as needed.	There is a large impact.
O2.2 Access to hardware platform engineers		
Engineers from the supplier of the hardware platform are available.	These engineers can act as consultants through our UE project.	There is a large impact on receiving regulatory agency certification of the operating system.
O2.3 Software design and training		
Half of the team has skills in structured design. For new algorithms, control panel and centralized data system, the staff must be trained.	It is feasible to mentor junior-level staff. Training engineers in these techniques is possible.	There is a low impact on meeting the schedule. However, there is a high cost to training all users of the system.
O2.4 Multi-process systems		
One fourth of the team is competent in these skills.	Training can be supplemented with software abstraction.	There is a large impact on all design choices.

O3: Schedule		
O3.1 Time and resources		
The UE organization is committed to dedicating the time and resources for a product line.	The business climate could change.	This could kill the project before launch.

Table 5. Organizational factors influencing the UE system.

2.1.4. Analyze Evolutionary Factors:

The Building Blocks for the Conceptual View

UE Evolutionary Factors

Evolutionary Factor	Flexibility and Changeability	Impact
E1: Software upgradability (system scalability)	Moderately flexible in Space if not too far; Highly flexible on Earth; highly flexible when PTVD-SHAM microchip is installed onboard or nearby orbiting satellites (see § 2.1) for upgrades; Automatically flexible in space based on the ABL characteristics	Main impact on hardware managers and controllers.
E2: Hardware upgradability	None in Space; Highly flexible on Earth.	There is a large impact on all components during development before Launch. N/A in space.

Table 6. Evolutionary factors influencing the UE system.

Only software-oriented upgradability is plausible for the travelling UE. Upgradability on the hardware-software, both, is plausible on Earth according to Figs. 1-3 engineering notations in use.

2.1.5. Develop Strategies:

The Building Blocks for All Views

Changes in Hardware Technology
Changes in the general-purpose and domain-specific hardware are anticipated on a regular basis. The goal is to reduce the effort and time involved in adapting the product to new hardware.
Influencing Factors

T3.1: Support is needed for uplinks and upgradability in Space as well as the station on Earth.
T3.3: The product must be ported in form of uplinks from Earth for the custom operating system, which could force a change in software platform.
P1 (Earth and Space): Projects interoperate and must support communication with external systems, to/from Earth/space. The system must be scalable enough and support simulation and distribution on Earth. In space, it must be flexible enough or adapt to the harsh conditions specified.
P3.1-3 (Space): The processor speed needs to be balanced against memory size and other factors to meet the product cost. This also involves signal compensators and other sensory systems with a memory structure to behave real-time performance relative to UE telemetry updates.

Solution
Encapsulate and separate hardware and software specifications.

Strategy: *Encapsulate general-purpose hardware.*
Encapsulating the system hardware allows advanced changes to be made to the hardware with little or no impact on the applications. This is provided by layers for the operating system and communication mechanisms.

Strategy: *Use a naming service.*
A distributed computing environment at the station and thus restricted to Space (not to a public network, and thus, highly local relative to operations performed in Space) via uplink and relaying information to the UE unit in space, is essential for distributing data over different types of processors. The goal is to be able to use a multi-core or many-core CPU, each core handling unique instructions (*reading* and *executing*) for a specific task e.g. compressing, acquiring, analyzing, etc. or some other combination. The central concept underlying the distributed computing environment is naming the service is naming the service relevant to the process type.

Strategy: *Partitioning and address bottlenecks.*
Developing the system to handle concurrent processes and/or threads requires delicate care in terms of balancing against memory size and restrictions. This is strategized in terms of compressing data to make all instructions available to the cores by a static information just using interpreters within the code structure. In other words, addressing bottlenecks by executing functions of some subroutine within the source code in binary, is possible when minimum amount of memory space is allocated with a valuable code combination (e.g. a translation table of some ASCII sort or **TT** file) is compared with other possible combinations to execute an explicit instruction. Therefore, the partitioning of data at any level is possible through compression and explicit access of this information, allowing more space for the CPU to maneuver and thus run other tasks, maintaining high performance.

Strategy: *Encapsulate device communication.*
Encapsulate details of communicating with devices into device-specific managers.

Changes in Software Technology

The software platform consisting of the UE RTOS, communication mechanism and data reconstruction methods (like image reconstruction based on the compressed binary data from Space) at the station on Earth, is likely to change when applications are ported or uplinked to the UE system (or even technological upgrades as new platforms at the station).

Influencing Factors

T3:

- Support is needed for uplinks and upgradability in Space as well as the station on Earth.
- Portability requirements are that the system must operate on different data types based on compressed binary data, operating systems and processors.

Solution
In addition to an operating system, add infrastructure software between the product applications and the hardware platform. This includes the restricted networking software communication mechanisms between Space and Earth, and interfaces, data management mechanisms, and a timer interface.

Strategy: ***Encapsulate operating system and communication mechanisms.***
Encapsulate operating system dependencies into an operating system library and IPC library that includes software timer support.

Strategy: ***Use reentrant repository for data sharing.***
Create reentrant repository that serves as dominant mechanism by which applications shared data no matter how compressed. We use related manager connections combined with asynchronous replies to data request as outlined in our execution bottleneck issue Fig. 4 (or see the last strategy made on Changes in Hardware Technology).

Usability by Target Audience

Projects are designed and implemented by UE engineers according to specifications written by process engineers. These process engineers also verify that a project design meets these specifications. The UE project engineering system must be easy to use for these domain experts. In addition, for the Earth station civil engineers have to meet these requirements as well as UE engineers designing the spacecraft before launch in terms of withstand radiation, spatial disturbances etc.

Influencing Factors
P.4.1-5 (Space and Earth):

- Projects must be portable to new hardware platforms for the station and the UE system before launch.
- The system must provide engineering tools for UE engineers to create projects, communicate results to process engineers, and perform verification easily.
- Projects must be impervious to disaster-related issues on Earth for the station as well as those encountered in space by the Satellite.
- The UE deployment projects must conform to regulatory efficient requirements for software quality.

P.1.5 (Space and Earth): The system must be scalable to support diverse project architectures.
P.1.3 and P.1.4: The system must support redundancy and fault-tolerance for projects relative to I/O frequent detections for the presence of devices onboard the satellite even in case of component failure.
P7.1 and P7.2: The cost to develop the project in terms of hardware must meet the spacecraft budgetary limit relative to latest technologies. However, software projects for verification, validation and license must not prohibit vital product pricing.

O2:

- People with a wide range of training are available as needed to support the project.
- The project engineers and UE engineers are not trained in classic software engineering analysis techniques as well as evolutionary software techniques like agent-based and AI.

Solution
Have domain experts (UE engineers, process engineers, electrical engineers, civil engineers, crises and destruction engineers, material physics and astrophysicists) set the basic software architecture concept for UE.

Strategy: ***Have the architecture designed by domain experts.***
The UE, process, and electrical engineers have extensive knowledge of domain specific notations that can be used to design projects. They are familiar with the variety of configurations these UE systems can support, and are familiar with their distributed nature. They understand the real-time, fault-tolerant, and

regulatory requirements of projects, and have experience building systems that fulfill these requirements. Destruction engineers and crises analysts shall evaluate the product during the development process as well as UE encounters in space as one crisis leading to another parallel/relative to any potential crisis occurring on Earth (e.g. disaster-related issues). All of these project characteristics have a profound impact on the design of UE.

Strategy: ***Generate code automatically.***
To generate code automatically, is primarily motivated by quality of software issue, but it also makes the system easier to use. The project designer is an expert in the UE domain, but is not necessarily an expert software engineer. With automatically generated code, the UE experts can concentrate on the project design instead of on the software engineering challenges. After launching the UE system, the adaptive codes onboard the craft could be studied by these experts on Earth for future counter-measurements in Space, thus providing uplinks or new software (additions) when deemed necessary.

Easy Addition of Features

Application policy rules change over time and are likely to become more complex in the future. The control panel as the monitoring system on Earth must present different features to the user, depending on the UE status, the connected UE peers between Space and Earth, and its connected I/O devices. On the other hand, the monitoring system in space as the operating system, processors and agent-based algorithms must maintain adaptability no matter how the range of complexity grows relative to space environment. These changes could occur during both monitoring systems run concurrently.

Influencing Factors

P1.1: There are thousands of customized settings and thousands of application policy rules.
E1: Software must be upgradable under non-real time and real-time circumstances for the UE unit.

Solution
Build applications from loosely coupled software components called entities. They communicate using a central repository and publish/subscribe protocol that provides immediate change notification. Each component owns or is responsible for publishing information (e.g., waveforms from space, alarms at the station) that is available to other components in the system. Other components can subscribe to the information.

Strategy: ***Use a central data manager and low-level repository to exchange information onboard the UE.***
Decouple applications by using a central data manager and repository to transfer information among the different applications. Information is exchanged much like a bulletin board using a translation key for the compressed binary code combinations, representing information or even instructions. On Earth, however, readable high-level monitoring system is required for humans but the same low-level information rules apply between the components at the stations' centralized repository system.

Strategy: ***Separate control panel and onboard monitoring functionality into loosely coupled components.***
Separate control panel functionality as well as the monitoring system onboard the UE, into software components that represent a logically-related set of application features. The application policy rules place requirements on the software components (entities) regarding the action, they perform, the information they need, and information they produce.

Strategy: ***Use publish/subscribe communication.***
Using data registration feature allows software component to ask for notification whenever data item changes. Software components do not poll. The monitoring systems' processing model is event or data-driven in almost all cases. All binary-based applications, they must dynamically accept and apply updates

to information, even while the agent-based system keeps monitoring/observing as a separate entity, executes preemptive communication protocols onboard the satellite (to avoid e.g. disaster from the external environment).

Quality Assurance

An embedded system is difficult to debug and test code.

Influencing Factors

O.2.3: The staff members have skills in building the monitoring system using the UE RTOS operating system for both Space and Earth.

T1.1 and T1.2: The processor speed is likely to change frequently, even during development. The number of processors may increase during development.

P3.1 (Earth and Space): Provide real-time presentation of the current monitoring state.

P3.2 (Earth and Space): Several processing priority levels are needed to support multiple processing deadlines.

P7.1 (Earth and Space): Trade of processor speed, memory size and other factors to meet the fixed product cost.

Solution

Place design constraints on software components to alleviate certain risks and highly-plausible bottleneck switches as discussed for Fig. 4. Provide support for diagnostics as newcomer data from Space.

Strategy: *Do not allow applications to share global memory directly*

All readable and compressed information shared among applications should be shared via inter-process messages or via the data manager. The replies to the data manager requests are asynchronous to prevent blocking among applications. The 2*n*-core inclusion assists significantly if the design is disseminated in form of hypercube parallel processes via a semaphore as a switching mechanism.

Strategy: *Create RAM-based error logging.*

Creating this, and an execution trace system to support debugging of target platform. The monitoring agent-based system should take-in significant errors relative to events as a future solution in rewriting exception-handling parts of the source code before a potential error occurs in the future.

Strategy: *Use static memory allocation*

The motive is that we need to avoid dynamic bottlenecks and data sorting which may increase algorithmic complexity (time) and use static or hardcoded solutions. Use static memory allocation to be less susceptible to memory leaks. The **TT** file LDC solution and the way fixed data compression ratio for incoming and outgoing data packets performed is a good strategy in addressing the problem. Additions like signal compensators and wave optimizers, as a memory solution, further mitigates restrictions on the memory allocation problem which entail significant external latency communication between Earth and Space.

Strategy: *Create tasks at start-up*

Create tasks at startup to avoid exception conditions with respect to resource management and task communication.

High Availability

Insuring high availability of the product is critical to its success.

Influencing Factors

P.4.1: Correct information must be displayed to the user at the station 24/7.

Solution
Implement mechanisms to alleviate certain risks (recall Bob's worst case scenario from §2.1).

Strategy: *Implement watchdog mechanisms.*
Implement hardware and software watchdog mechanisms to ensure software is not in an infinite loop (deadlock).

Strategy: *Implement the recovery mechanisms.*
Implement recovery mechanisms in the data manager to ensure the integrity of the data in the repository, even if the station power on Earth was lost during data write operation. This also includes backup systems onboard the UE in case of power loss, etc. specified throughout §1.2.

High Performance

Meeting the real-time performance requirements is critical to the success of the product.

Influencing Factors

O.2.1: Correct information must be displayed to the user at the station 24/7.

Solution
Use 2*n*-core processors and real-time operating system to meet the high performance requirements. Prepare for different processor configurations in the future. Partitioning the system into separate components gives flexibility in allocating them to different processes. Compressing the system resources with fixed results (fixed allocation) with a reliable code combination key (fixed as well) to save space and time for the processor to execute more tasks.

Strategy: *Use 2n-core processors.*
Use a digital signal processor for signal filtering, use the fastest custom-purpose processor available for main processing, and waveform drawing, other data type reconstruction to a graphics processor after data decompression at the station (Earth). The 2*n*-core inclusion assists significantly if the design is disseminated in form of hypercube parallel processes via a semaphore as a switching mechanism.

Strategy: *Use the UE RTOS operating system.*
This real-time operating system, has a strict priority scheduler, supports message sending and receiving, and has semaphores explained previously. These are typical RTOS system features.

Strategy: *Divide logical processing into multiple components.*
Divide logical processing into software components, called entities, to meet timing deadlines.

Strategy: *Separate steady-state real-time processing from event-driven asynchronous processing.*
This separation with critical real-time performance requirements from those with less stringent requirements.

Strategy: *Create a task for each unique processing deadline.*
Use rate *multichannel monotonic analysis* (incoming data from all sensors to the processors) for assigning software components to tasks based on processing deadlines, and to give higher priority to tasks that run more frequently or have a shorter deadline.

Strategy: *Do not allow entities to make blocking calls.*
This is reiterated in terms of addressing bottleneck issues relative to decoupling request from replies into separate messages.

Strategy: ***Minimize the copying of data.***
Analyze the impact of data copying and eliminate it when necessary as a filter solution on the BDB component. Consider using a shared memory queue for high-volume data when appropriate.

Strategy: ***Don't produce data unless it is needed.***
This filter solution is necessary as explained in terms of those data that occupy void or unnecessary space (see, Fig. 1, Step #3). Write high-volume data (e.g., waveforms) to the data repository only if the wave represents rich data from space (non-empty or void).

More on the Evolution or E1:

Software upgrades must be possible in aim of bypassing managers and controllers, or instructions in case of errors by an intelligent based system, as well as previously-encountered experiences in space to avoid mistakes.

Solution
Use software and code agent-based learning algorithms independent of the 2n-core processors, which also shouldn't affect current performance. In most cases, such algorithms bypass managers and controllers by rewriting parts of a code that execute functions in a traditional fashion, now converted to short execution paths as a solution to an error or crisis which is about to happen.

Strategy: ***Produce target-independent code*****.**
Keep the code that implements the UE system independent of the target system platform. Although the project designer assigns CPU number specifying where each part of the UE functionality executes (like other hardware components), the resulting code should be independent of the particular CPU type. This also entails the evolutionary programming features of the growing code based on observing errors detection and whatever being experienced by the components aboard the satellite. It also learns how to evolve and anticipate the rerouting strategies on rewriting the previous code independent of the target system platform which results in bypassing the device managers and controllers of the system as a solution.

2.1.6. Quality Requirements:

Software and Hardware

The following list addresses the UE software/hardware quality requirements, based on software quality evaluation model ISO/EIC 9126-1 [38] for its components:

QR1. The system must be *fault-tolerant*, and in case of any hardware or software component failure, it must continue functioning.
QR2. The system must avoid any data loss even in case of failures (QR1). The system should be able to trace the last actions and errors performed before system failure.
QR3. The system should be able to recover itself after an unexpected failure or breakdown. When it runs again it should go back to the same state it was before the failure.
QR4. The system must be able to avoid loss of data and react to failures, potentially using backed-up data in case of disaster-related issues. We **strategized** in terms of uploading the station's compressed data to home-orbit secondary satellites, downloading all information to a second station which could be in any other geographical location on Earth. The signal bandwidth limit

is not of significance, since we use the latest LDC algorithm for ultimate compression ratios at the station, however, on-the spot transmission of data to these satellites depends on the satellite tracking program (recall step # 5, Fig. 3) for a trustable upload/download to the relative base on Earth.

QR5. Failed hardware component should also be recoverable or replaceable when needed via software agents relative to controllers e.g., CPUs as a backup solution.

QR6. The system should adapt to its space environment, the physics in terms of updated/uplinked telemetry by the either artificial or human operator, using smart and efficient algorithms

QR7. The internal system should also adapt with its newly assigned on-board communication protocols in aim of controllability and interoperability parallel to any potential internal/external errors (e.g. sensors, see also Code View on the ABL learning algorithm or §§ 6.1.1 and 6.1.2)

QR8. The system should evolve based on previous mistakes, if they potentially attempt to reiterate, the agent-based learner (ABL) algorithm must be mature enough to address this potential threat and avoid similar mistakes in the system's log.

QR9. The system should be stable enough to maintain all onboard operations between hardware components relative to external information updates (either in space or from Earth).

QR10. The system should be tested for all of its components, hardware and software before launch. The system also must be testable after launch in space via Earth station communication units. (This includes software upgrades from Earth to the UE system)

QR11. Onboard sensory and processing agents, algorithms, etc. must be able to analyze incoming data from space or Earth, and any asynchronous communication should be normalized down to ≈ 0s real-time scenarios.

QR12. Before information uplink, concerning manual telemetry updates to the satellite, systems at the station, should be able to simulate new information (via simulation pad and DBs), based on analyzing the previously-incoming data from the UE system.

QR13. The system on Earth must be able to run under new hardware resources that allow it to improve in performance. The UE in space must continue functioning with new software updates.

QR14. The system's incoming and outgoing information must be secure enough between Earth and space. This is **strategized** in terms of compressed information which is exponentially impossible to decrypt the message with a decompression/decryption key on Earth, where the key gets updated on a daily basis. The latter key is *dynamic* as a superimposed layer to the *static* key attached for decompression.

QR15. The system must support and handle concurrent access, giving priority to remote relative to local request executions, from Earth to Space. This, both involves interoperability and accuracy sub-characteristics of functionality.

QR16. The executions within the UE system in space must be as efficient as possible and avoid latency issues when relaying data to Earth. This entails imposed physics latencies as an external data relay barrier. (We have further **strategized** in terms of a waveform tuner/accelerator or real-time compensator as a memory-sensory component onboard the system or PTVD-SHAM [23].)

QR17. No latency issues must be visible at the station when operating and updating the satellite from Earth.

QR18. The system must be able to access vital data at any time either onboard or from the station on Earth. This is **strategized** in form of a centralized data system with relevant communication units to uplink vital data to the satellite.

QR19. The operators at the station must be confident enough to operate the system remotely based on their previous trainings. Therefore, the interfaces must comply with the usability attributes such as learnability, comprehensibility for its operators. This entails operability between functions available to the user at the control panel as well as other compartments of the station.

QR20. The system must be scalable enough, at least in software products, and be able to run in a multi-core processing environment, taking advantage of the available resources relative to information load (addressing bottleneck issues).

QR21. The system must support easy additions of functionalities and enhancements (software upgrades).

QR22. To provide reliability at the station for on-time communication, the system must support replication on the centralized databases, their partitions, etc. to be able to react properly in the case of failure of one of the databases, even the one installed onboard the UE satellite.

QR23. **T**he system must display correct information on Earth, 24 hours a day, and 7 days a week (P.4.1).

QR24. The UE satellite must communicate with Earth (UE station) under harsh conditions in space (P.1.3, P.4.2), while onboard intercommunications between components monitored, processed and managed.

QR25. The UE system must startup, self test components before launch, and must be able to reboot safely in case of major failures in space, it must also be able to conduct the same or even updated routines by its custom *real-time operating system*, or UE RTOS. Note that during any updates, the OS must not alter its real-time performance (fixed deadlines) and its multitasking assignments. We **strategize** to use only compressed binary data with a readable static key onboard the satellite, thus for its newly-assigned tasks as updates on a particular application, this solution would not overwhelm the system with extra information flow and memory usage.

The abstract version of our quality requirements is listed below and specified (labeled) in the general scheme of our architectural topology, Fig. 6 , which reflects Figs. 1-3 from §1 of the report.

1- Reliability relative to Availability
- 1-1- Fault-tolerance (Component Failure)
- 1-2- Recoverability
- 1-3- Maturity (ABL-related that avoids failure as a result of faults in the software)
- 1-4- Reliability Compliance (Confidence)

2- Functionality
- 2-1- Security
- 2-2- Interoperability
- 2-3- Accuracy

3- Maintainability
- 3-1- Stability
- 3-2- Testability
- 3-3- Analyzability (Agents and Algorithms)
- 3-4- Changeability (Agents and Algorithms)

4- Portability
- 4-1- Adaptability (ABL-related)
- 4-2- Installability
- 4-3- Co-Existence (Memory, Algorithms + Agents)
- 4-4- Replaceability

5- Efficiency
- 5-1- Time Behavior
- 5-2- Resource Utilization (UE's Centralized Data System on Earth)

6- Usability
- 6-1- Learnability
- 6-2- Comprehensibility
- 6-3- Operability

Quality Requirement	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Availability	X		X												X								X	X	X
Reliability	X	X	X	X	X			X						X								X	X	X	X
Functionality						X	X	X		X		X		X	X	X		X			X				X
Maintainability		X				X		X	X	X	X	X	X				X			X	X			X	X
Portability						X	X	X				X	X					X			X				X
Efficiency											X	X	X			X	X	X				X		X	X
Usability												X							X						X
Integrity		X	X	X										X										X	X
Scalability								X	X		X	X			X		X	X		X	X				X

Table 7. Quality factors influencing UE and its CCS systems.

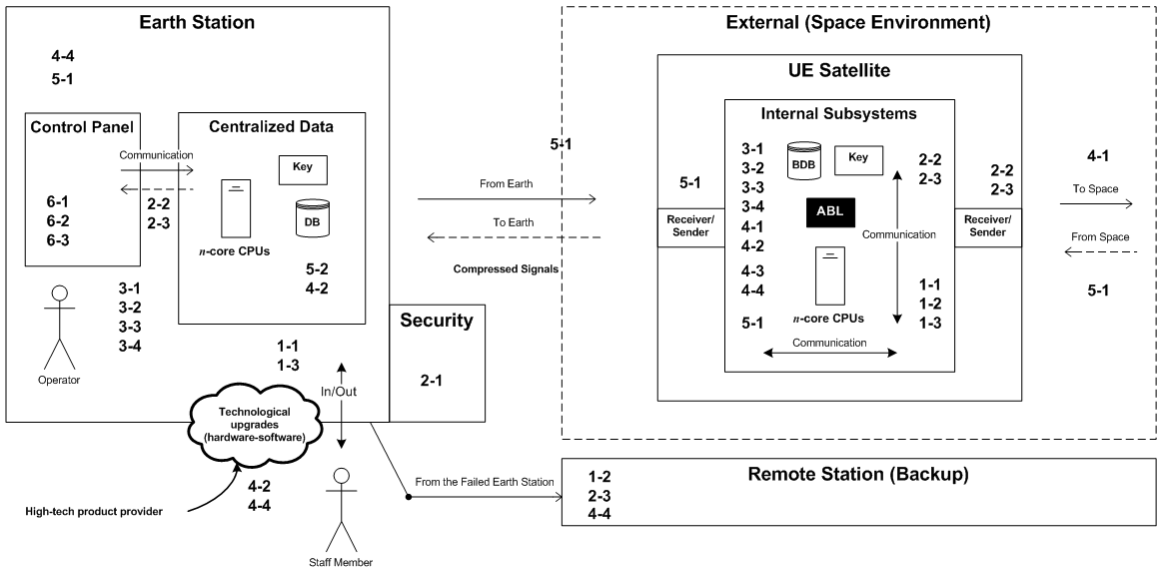

Fig. 6. The simplified overall topology of Figs. 1-3 with quality requirements labeled in Space and on Earth

2.1.7. Architecture Transformation before an in-Depth View

A High-level representation: ***A refined transformation stratifying view layers on our design choice relevant to quality requirements via strategies***

A High-level Representation of the UE Satellite Software Design:

The simplified classification of the concrete version of all software architecture views are give as following with notice to the merits of Fig. AT1:

UE Data View:
The top most level of the application is the operator/user interface view incoming info from UE in space. The main function of the interface is to translate tasks and results to something the operator/user can understand.

UE Data Update:
This level coordinates with the application, processes commands, make logical decisions and evaluations, perform calculations and communicate with different devices attached. It also moves and process data between the two surrounding levels of view and updates, in space and one Earth.

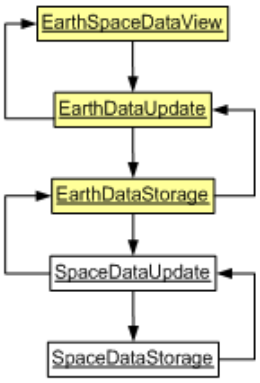

Fig. AT1. An initial high-level configuration of IS2000

UE Data Storage:
Here, the gathered information by the satellite is stored and retrieved from the database. The information is been transferred according to the requests made from within the system or across the satellite network.

Refining the Conceptual Relative to Other Views as a High-level Design Representation

The following figure is a refinement of the decomposition of our views mainly Conceptual relative to the performance related issues as well as strategies pertinent to the remaining software quality sub-characteristics:

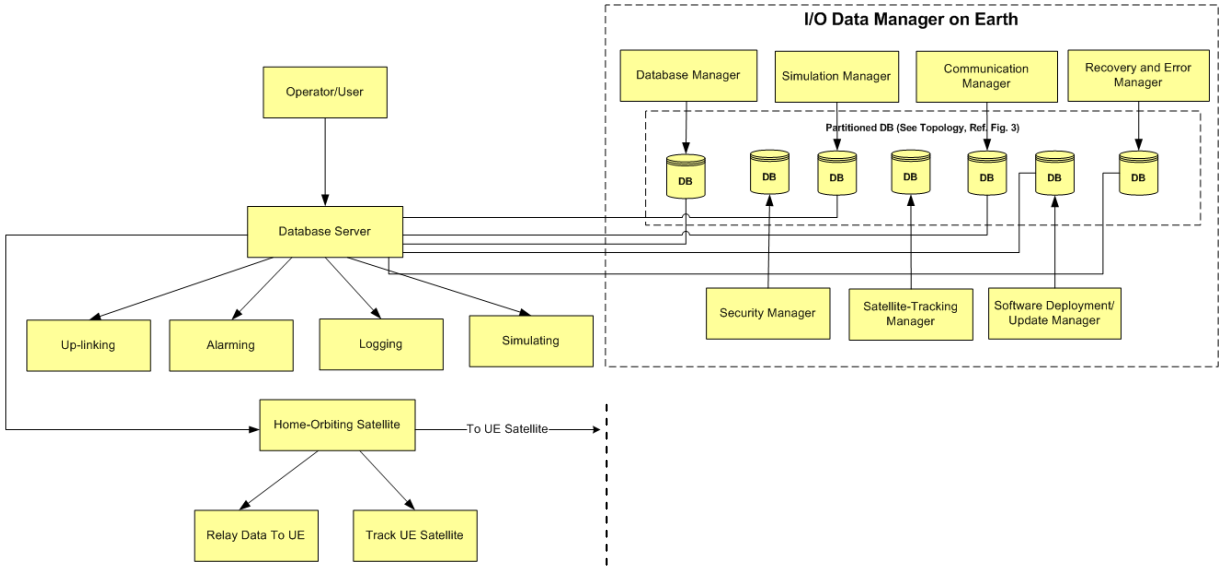

Fig. AT2. This figure represents an Architecture Transformation upwards to a concrete high-level design on software application and its relative components at the station on Earth. Data is accessed by the operator, thereby uplinked as a

simulated new coordinates to avoid crisis on the satellite unit, as well as a software update on any hardware-software control based on the partitioned database: DBs, here, are in fact partitions. The partitions represent each manager on Earth.

Fault Potential Problem on the Early Evaluation, Rectified

A potential storage problem would have sustained initially, if we would have considered an ordinary DB storage technique on data. This resulted in our choice to compress data and thus manage space and allocate process time between components.

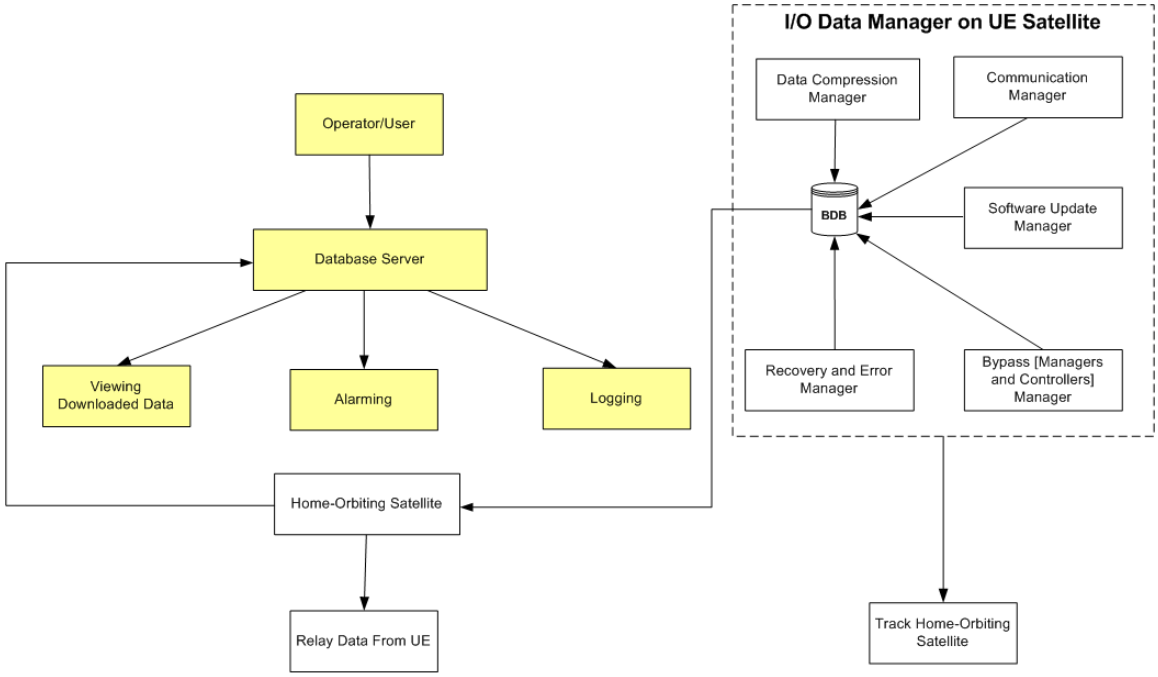

Fig. AT3. In this figure, the main database component is a binary database (BDB) and supplies all UE data managers onboard the UE, *the information needed via a data translation key index combinations* for its compressed version. After accessing this data representing, virtually, the massive amount of information, obviously, system components are managed by the application for specific tasks.

On the multiprocessors problem, that flaw potential is too contemplated in the first place, Fig. 4, §1.2.2, using the software bottleneck partitioning and semaphores at the *data application updates* recognizable in Fig. AT.1, As we recall in that section, this evaluation part of our architecture was submitted as a solution to our Execution View addressing bottleneck against usual architectural techniques used in the software application layer (recall Fig. 4). The time-length reduction or decrease against any potential time-length extension or increase, has been technically discussed and refined in our Execution View. Of course, this relationship between the BDB component and processors is significant on *all views* when Space Data is packetized and sent to Earth for view and real-time updates. This conversely is the same when Erath updates space data by sending new coordinates or instructions to command the satellite under critical conditions (e.g. Bob's scenario, §2.1).

Of course, further strategies expressing our data managers are already presented in our Module View in the decomposition of the platform software, §4.1.1.

The Refined Transformation Steps:

1. Satellite Data Updates and performance

In order to achieve the performance requirement (QR.6 and 9) the vital data operating the satellite (current and future code updates) that are not stored on the local server are requested to the Index Server that sends the request to the server at the station, where the UE updated data is stored, so this one sends the data by packages sorted by a given criteria and the complete code is stored in the requesting server where the changes (if possible by permissions) are stored onboard the satellite in a compressed form. This characteristic permits to access the satellite within the least time-delay possible from the control panel at the station that is updating it or consulting it. The detailed way of the information is sent between UE and Earth station and obeys a given criteria, so the UE data that is received and can be shown while the rest of the information is sent and stored. To achieve maximum performance the component identifies the read and update requests. All the read requests are handed over to Index Request Handler which is closer to our Database component; this results in easy and quick handling of request and usage of less system resources, especially on the satellite as compressed data. The Component of Update Manager as well as Database Manager, both have these roles on repository index updates and information access. The communication is in form of data packets as purely compressed data to the satellite and form the satellite instituted by the Communication Managers in Figs. AT2 and AT3. The index update is explicitly relevant with controller components shown later in the Decomposition of the DB managers, Figs. CnV6 and CnV7.

i. Device communication

The communication between the UE system and different devices is handled and encapsulated by the Data Control Unit component. The device control unit checks the availability and platform of communication for the devices to communicate directly to the System or system(s) between Earth and space. Data is compensated and manipulated when necessary in terms of compressed data during communication between the satellite and the base on Earth. The compression is explicit in terms of integrity and thus no data loss at the decompression phase on Erath for its viewer/operator. Compression is essential for efficient communication relative to data compensators which depend on the absolute physics restricting real-time communication from/to space (subject of Execution View). Without considering this technological factor, delay of data communication is inevitable. This Control Unit component sends the respective commands to the specific device, fetches the results sent by it and sends back to the Update Manager in a form of Data for its further processing, when ever acquired by the satellite in space. This manager is part of the I/O management system on UE as illustrated in Fig. AT2. This is also applicable on Earth when the control panel operator acquires data from Space for further processing via a similar approach, (Fig. AT3) delivering results as decompressed data. As stated before, the controller component is abstracted with Ctrl in the later figures of our Conceptual View (the conceptual configuration figures).

ii. Recovering after failure

In order to provide recover mechanisms for the system after failures, a special process for tracking the system transactions, log every activity and to register the proper turning off of the system or in future cases, when system learns, directly fix it while error is occurred or system failure. In case of an unexpected shut-down, when the system starts again it will run the start-up process to check the compressed data representing specific error codes or type in order to detect if there was a proper shut-down of it was unexpected. In case of detecting an unexpected shut-down the system will run a recovery process responsible to track all the actions previous to the failure that have been registered in the logs file. After recognizing the actions that were not executed because of the failure had been recognized, the system will book them to re-start again. The compressed database engine will protect the system from the negative effects of interrupted transactions by its Error and Recovery Manager (Fig. AT3). The system will also give a notification to the operator at the control panel on Earth that start session after having interrupted

transactions or system failure to notify them the last transaction performed, so ensure the UE system has restarted at the last saving point otherwise, all restarting procedures must take place by up-linking the necessary codes and instructions from Earth to the satellite (Addressed in our Code View). All this is possible because of the system log process running constantly to trace all the transactions performed. This is shown in the "Logger" code view process in §6.1.2.

iii. Data Safety

To provide safety to the stored data in the database at the station, a local backup database has been designated to work as a mirror of the primary DB in form of a separate partition or even server elsewhere (formerly designed in our Topology, Fig. 3) which is handled by the database server. This partitioned database allows communication and data control to act efficiently and address bottleneck issues resulted from the load of incoming data from the UE in form packets of bits as they accumulate during real-time access. In case of the failure of the primary database the mirror database will be available to support the system operations. In that case a general backup to the main server will start and a warning will be given to the administrator or operator who will decide in what conditions the system will continue. If any of the primary or secondary database partitions fails, the backup will be made on remote location before shifting to available functioning database (UE backup station from Fig. 3 adapted to a recovery manager criterion during operations in Fig. AT3). Two types of backups are made one locally managed by secondary database and other is done remotely. Local backup is made according to each update made on database and remote backup twice or thrice a week according to UE station policies.

iv. Security

The secure access to the system is handled by the authentication/authorization unit component, and this functionality is mapped to the UserAccessLevelProcessing module in the security layer and the overriding or Bypass Manager (in case of critical) in the process running at the local server.

2. Data Manager

Data manager will manage the following things as mentioned below based on Figs. AT2 and AT3.

a. Interface (at the control panel on Earth)

This is the Operator's Interface with all the different controls which the user sees and uses to access functionality of the system.

b. Validation (in Space and on Earth)

This is used to perform some satellite side validation of telemetry data updates and addressing crisis management issues in space (Recall Bob's scenario from §2.1) like checking the format of correct data on real-time or delayed time, sending off the correct coordinates, software updates or checking of the required backup systems or an ongoing functional hardware during software operations.

c. Attachment Unit

It's a specialized interface for easy and quick viewing and attaching different copies of historical records or even code updates including different ongoing machine tests result.

d. Process Unit

Process unit is used to handle the movement of data across different units. The data refining and their transfer is been handled by this unit.

3. Output Controller

It is responsible for displaying operator's requested information at the station on Earth. All ongoing updates are managed and notified to operators in case if any other operator/user is accessing old information. Aboard the UE, this is managed by bypass and communication managers sending information to Earth as well as instruction access and updates by the main processors for controlling the satellite.

5. Update Manager

The following things are managed under Update manager as given below.

a. Lossless Data Compression Controller

This component manipulates the data into smaller volumes by using state of the art encryption algorithms (specifically with fixed ratios to make data management quite efficient) and other techniques to guarantee the losslessness of data between Space and Earth. This falls in the DB manager's area of operations on the low-level repository: subject of the strategy "*Use a central data manager and low-level repository to exchange information onboard the UE*," from easy addition of features.

b. Authentication/Authorization Unit

This component is used to make sure that the operator on Earth accessing the system has authorized rights to access general and specific information according to their user rights and hierarchy with overriding security features (from Bob's scenario, §2.1).

c. UE I/O Data Management Unit

It's used to manage information movement between different components and controls the fellow of data between different devices within the UE system.

d. Compressed and Decompressed UE Content Manager

This Content Manager identifies the type of action to be performed on the data and pass it to their respective components for processing. And finally update it into the UE system.

e. Update Manager

This component is used to handle all activities related to modification or retrieval of data from the system and pass the information back to Compression and Decompression UE Content Manager.

f. History/log Manager

It keeps the log of all the activities undergoing in the system.

g. Satellite Data Sharing

All operations related to satellite movement from one space coordinates to another or any updates onboard the satellite are performed by this component which depend on the compressed DB component translations (data interpretation by Software Code in program source aboard).

2. Device Control Unit

Device Control Unit makes it logically possible to communicate between different devices connected with the UE system, and it manages the flow of Data between system and device.

3. Storage Management

Data Access Logic
This component handle the functionally for accessing the data in the database like how to add, delete, update information in the data in compressed manner just relying on shortest bit combinations rather than high-level data onboard the UE. High level information is essential on Earth when manual inputs by the user are concerned.

Data Optimizer and/or Compensator
Data Optimizer and/or Compensator are used to achieve maximum efficiency related to data compression management, storage capacity, transmission time etc. during updates and instructions. This involves Bypass Managers both in Space and on Earth to bypass other mangers and controllers when necessary, satisfying shortest execution path (route) possible to either send or receive data.

Information Unit
Its task is to evaluate the type of update and notify accordingly. This can be a physical data update or just a simple update notification.

Storage Units
Both on Earth and Satellite, it is responsible for all kinds of activities which are performed on the compressed database, and thus to make sure the data has been modified when deemed necessary through program instructions (performed by processors mainly detailed in Execution and Code Architecture views).

Software Update management
Onboard the UE, the software update manager uses the BDB component to allow necessary updates occur, to assign tasks to the evolved entity (ABL) component managing errors as well as direct management of hardware components. The evolved entity is introduced in our Conceptual View and further specified on its roles and evolutionary characteristics in Code View which address the evolutionary E1 strategy of the system.

Bypass managers and controllers management

Onboard the UE, the Bypass Manager uses the BDB component to allow a bypass of certain controllers to issue high performance strategy as well as data recurrence or redundancy: Produce target-independent code from evolutionary factor E1. Furthermore, bypassing controllers when identical data is processed or satellite reappearing in the same space coordinates, or any other issue increasing time delay such as recovery mechanisms, this bypass compensation factor is essential. This is later considered in Figs. CnV4 and V5 in our conceptual view as green-zone controllers in the view's configuration, which addresses the two strategies: *Implement watchdog mechanisms* and *Implement the recovery mechanisms* from *High Availability*, as well as performance-related strategies in our software quality sub-characteristics.

Data Compression Manager
It insures that the compressed data represents the right instruction codes on managing hardware components as well as software updates.

4. Index Communicator

All remote update and notifications are managed by this component and the decomposition of it presented in the Conceptual View as Figs. CnV6 and V7, in §3.

Summary of transformation

The results of the evaluation are shown in the Architecture description before the views (§§1 and 2) defining our choice of design. Since the new design choices were made from those points inclusive of an early evaluation which was not fully given by the peer-reviewing group, the initial strategy was to consider all necessities of the design in the first place before any transformation. Furthermore, the evaluation led to a second evaluation as the results of the first evaluation in terms of the Code View based on the evolved entities introduced in the conceptual.

We remain to emphasize on the self-evaluation of the architecture both in Space and on Earth on the conceptual, since after defining our entities, the applicability would be already defined in the Code Architecture view (§6.1.3) and its early core evaluation (Section 1.2.2) on the remaining architectures indeed. The latter emphasis is mainly on what we have evaluated for our software solution addressing bottlenecks (Execution Viewpoint and *software resource management*) which involved the asynchronous parallel processing solution (§1.2.2 or Fig. 4) and compressed binary database components for our data mangers (this was considered initially).

All of which we have already incorporated in our Conceptual as well as Execution and Module views both on the Earth and in space data managers i.e. for a successful data transmission and its subsequent management: *points of data analysis from a human operating at the control panel to the machine levels as the central data on Earth (elicited form the early sections of the report topologies) and satellite in space, process, control and delivery*.

The following is the configuration and decomposition of the main system on all views relevant to the current software architecture and quality topic.

3. Conceptual Architecture View

3.1.1. Components for Software Specification:

Function Blocks, Connectors and Categorization

Category	Examples of Function Blocks
Arithmetic operations on analog signals	add, sqrt, abs
Analog signal processing	Unit delay, differentiation, integrator, controller, digital filter
Selection of analog signals	Switches, min/max value selection, sorting
Mixed analog/binary signal processing	Limit switch
Logical operations	and, or, xor, not
Binary/fuzzy binary/quantum signal processing	**Internal/onboard circuitry:** Unit delay, pulse with specified duration, on and off delay, flip-flops **External/UE to Earth:** Space-time delay, pulse with specified duration, pulse bender/accelerator/superposer or time-data entanglement via PTVD-SHAM memory
Selection of binary/fuzzy binary/quantum signals	Switches, *n*-out-of-2*n*/four-out-of-four/three-out-of-four/two-out-of-four/one-out-of- two states voting (the *n* out-of-2*n* is unique to the many-core hypercube processors parallel to

	PTVD-SHAM component: select incoming signals from Earth, boost or accelerate outgoing signals to Earth)
Domain-specific actuator interfaces	Interfaces to control pneumonic components for navigation, depressurization, heating elements, blower, wing system, CNT physical properties, extinguisher, and shield
Interface to the runtime environment onboard	Acquire operation-mode permission, pass error flags against AWK signals (see, Code View)
Interpret compressed fuzzy binary signals into binary signals onboard	Acquire instruction mode between hardware components and CPUs by a fixed combination of binaries from the *C*(DB) component, which has a fixed **TT**able satisfying these combinations.
Interface to the runtime message dispatch	Send finalized processed signals to Earth as an external approach via e.g., the PTVD-SHAM component
Relay dispatched message to Earth	Acquire passive mode message from UE, relay to station via, e.g., home-orbit satellite
Decode compressed message signals into binary signals on Earth	One for each message block module that generates a message signal after decompression via its database equipped with the TT key at the Earth station.

Table 8. Categorization of Function Blocks onboard relative to station on Earth

In some scenarios, we have abstracted the function blocks (FBs) for our conceptual view to satisfy a comprehensive conceptual scenario and configuration. For instance, the *miscellaneous physics*, which is to do with all physical components (hardware) onboard the satellite including the shield system, sensory systems, change of CNT physical characteristics, etc., includes such function blocks like the actuating interfaces example from Table 8, controlled by *software instructions* passed from one main CPU component to another, in concept. We have encapsulated the very details of the detailed information transacted between components (communication) via entities, some as *agents* relative to non-*agents*. The software component as a device manager presenting an abstraction of a UE device, applies the strategy of *encapsulated device communication*, from § 2.1.5.

In continue, we shall experience *asynchronous communications*, which is due to using connectors for indirect communication between entities addressing high performance issues onboard the UE. However, the FBs are specialized for *direct communication* at the CPU level satisfying near 0s time scenarios. Therefore, in our conceptual view, we have these FBs, each used as a source (**S**) and destination (**D**), connecting FB_ports to FB_roles, where asynchronous communication takes place, and in parallel, all of the time cycles are normalized. These roles are classed for either *incoming raw data* or *outgoing data* onboard the UE.

At the execution level, the choice of connectors from the conceptual reflect in the connected ports to roles, when communications occur at the CPU level/memory, which should fulfill real-time scenarios, already rationalized back in § 1.2.2 during our system evaluation.

We further refer to these examples in aim of proper terminology and notational use, as well as relevant examples in our other views.

3.1.2. Sender/receiver Processing Scenarios and Conceptual Configuration

Processing Scenario, Meta-Model, UEDOMessage Protocol and Configuration

The scenario in Fig. CnV1, uses a UML Collaboration Diagram (see Chap. 9.2, Hofmeiester [1]), to show how raw data from space is acquired, processed, and displayed on the Control Panel screen on Earth (the main station or base). Before displaying information to the user at the control panel, the UE system device managers and controllers, onboard the UE, decide real-time what to do for critical and regular situations. It is either receiving interruption request signals (IRQ) from hardware components, and thereby acknowledgement signals (AWK) to components via the main CPU, otherwise, compensate/recover through backup or take care of a particular situation in terms of avoiding dangerous or critical scenarios in space. The *waveform data* (wvf) under any of these conditions shall be relayed to Earth, i.e., control panel, and further analyzed, processed and ergo decided upon for upgrades and human-based decision plans/executions.

Although these scenarios do not give the complete picture of conceptual view, they do show the peer-to-peer communication that is missing in the module view.

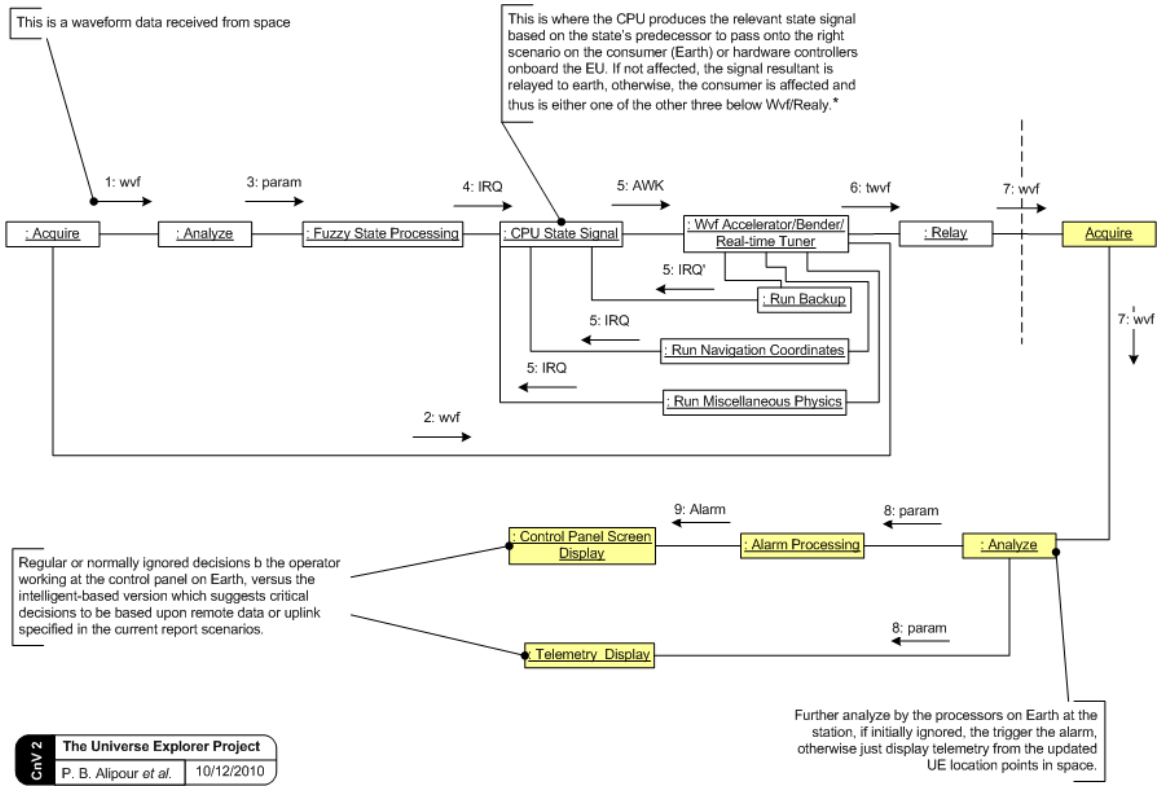

Fig. CnV1. Receiver/producer/compensator/accelerator and relay processing scenario from Space-to-Earth

The software components in the scenarios are a particular type of component called a software entity. In the conceptual view, the entity concept is the result of applying the following two strategies:

1. *Separate monitoring system functionality into loosely coupled components* (issue Easy Addition of Features)
2. *Divide logical processing into multiple components* (issue High Performance)

A software entity produces data that is used by other entities. Thus, it is possible to describe the conceptual view of the UE system using software entities a s the lowest level components. Fig. CnV1, shows that an entity communicates with other entities through its ports via messages (a UE global data object or [UEDOMessage]). Some of the message types for the monitoring entities are the raw data coming-in from the devices, waveform samples, parameter values, and *signal state status* (in case of Earth, at the control panel is the alarm status) to vote from a fuzzy state, binary, etc. relative to a particular UE hardware component . Each message type is produced by only one entity, but many entities can consume it. Entities are decoupled in the sense that producers don't know which entities consume its data and consumers don't know which entity produces the data.

The meta-model in Fig. CnV2, is a restricted version of the conceptual view meta-model presented in Fig. 4.16 of Hofmeiester [1]. The most important simplification is that UE has only one level of decomposition; neither the components (Entities) nor the connectors (UEDOConnector) can be decomposed. A UEDOConnector has exactly two roles – a producer and consumer – and this can only be connected to output ports and input ports, respectively. Because the UEDOConnector cannot be decomposed, the protocol, (UDOMessage) is associated with the connector instead of with its roles. And lastly, because an entity cannot be decomposed, there is no need for bindings between ports.

The conceptual view can also explicitly describe abstract connections between entities by describing the protocol associated with a UEDOConnector. It contains information about the characteristics of the data and events flowing between the entities, including their type, direction, size, and rate. The strategy *Use published/subscribed communication* (issue Easy Addition of Features) applies to this protocol.

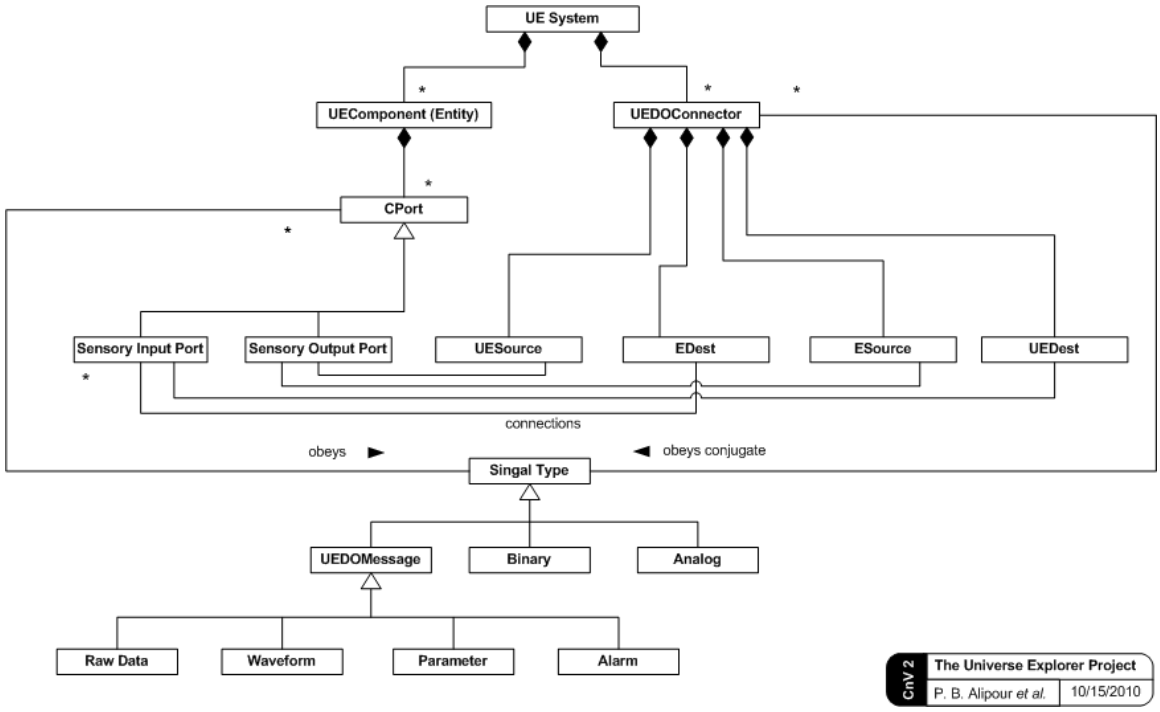

Fig. CnV2. Meta-model for UE entities. ESource = Source on Earth, UEDest = UE destination in Space, UESource = UE Source in Space, EDest = destination on Earth.

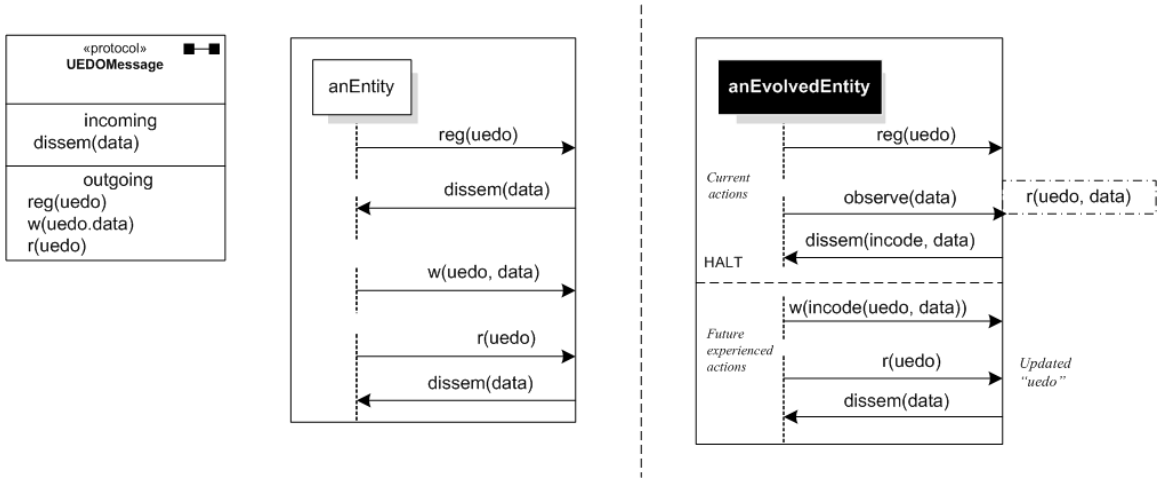

Fig. CnV3. UEDOMessage protocol on the normal Entity adjacent to an "Evolved Entity", where the latter is exemplified in terms of an agent-based learning (ABL) algorithm in our Code View, which updates communication, state control, or even upgrades system applications' code based on previous UE experiences in space

The protocols adhered to by entities communicating via the UEDOConnector is shown in Fig. CnV3. On the left image, the information exchanged between entities via the UEDOConnector is represented as UEDO. Entities first register for the data of interest. When published, data is written, it is disseminated to all entities subscribed to it. An entity can also explicitly read data rather than wait to be notified of an update. This could be recognized on the right image which is observing data (or read) then disseminating data in the code's body for future communicational usage. This is an Evolved Entity that could update the level of communication between other entities when necessary. Therefore, in its future calls, it uses the updated UEDO and disseminated it to all entities subscribed to the current data, which is by now updated via this entity (the evolved type). A good example is an agent-based learning (ABL) algorithm which is later discussed in our Code Architecture view.

There is a special kind of entity called an *agent* like the last example, which acts like an intermediary between other (non-*agent*) entities and device managers. A device manager is a software component that interacts directly with hardware device, and presents to agents an abstraction of this device. The device managers are the result of applying the strategy *Encapsulate device communication* (issue Changes in Hardware).

Fig. CnV4, shows a portion of the conceptual configuration for UE both in space and on Earth (the ones on Earth are colored in yellow). This configuration corresponds to the scenario depicted in Fig. CnV1, but here the input and the output device managers are also shown. There are four types of UEDOConnectors in this configuration: WUEDO, PUEDO, SUEDO and AUEDO. These connectors handle the communication between entities and convey waveforms, parameter, state signal onboard the UE, bypass opportunity in case of identical incoming waveforms to their past, and alarm data at the station. Connectors of type UESM and CPDM handle the communication between the system device managers on UE with respect to device managers by the control panel. There is a special OptimzedWvF connector as [filtered + fragmentedWvF] (of partitioned type) for the high-speed data transfer used to deliver the waveforms relatively real-time to Earth. The bypass opportunity (the green zone conditional connector bypassing device managers between the acquisition and relay) is conditional to an agent-based learning algorithm which addresses issues related to high-performance on selectable incoming waves for short path communications (later elaborated in § 5.1.2 of our execution view).

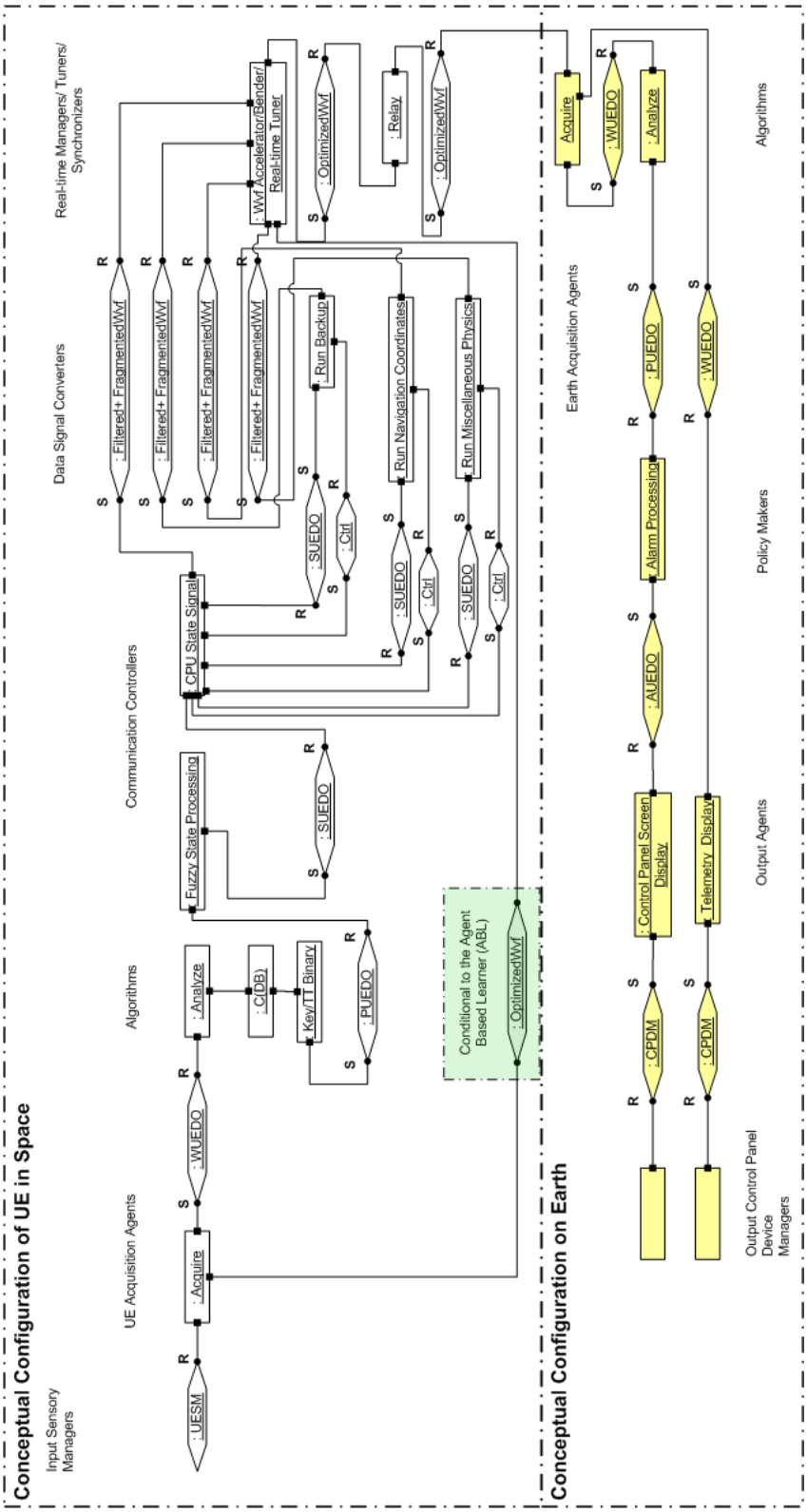

Fig. CnV4. Conceptual Configuration of the UE system between Space and Earth

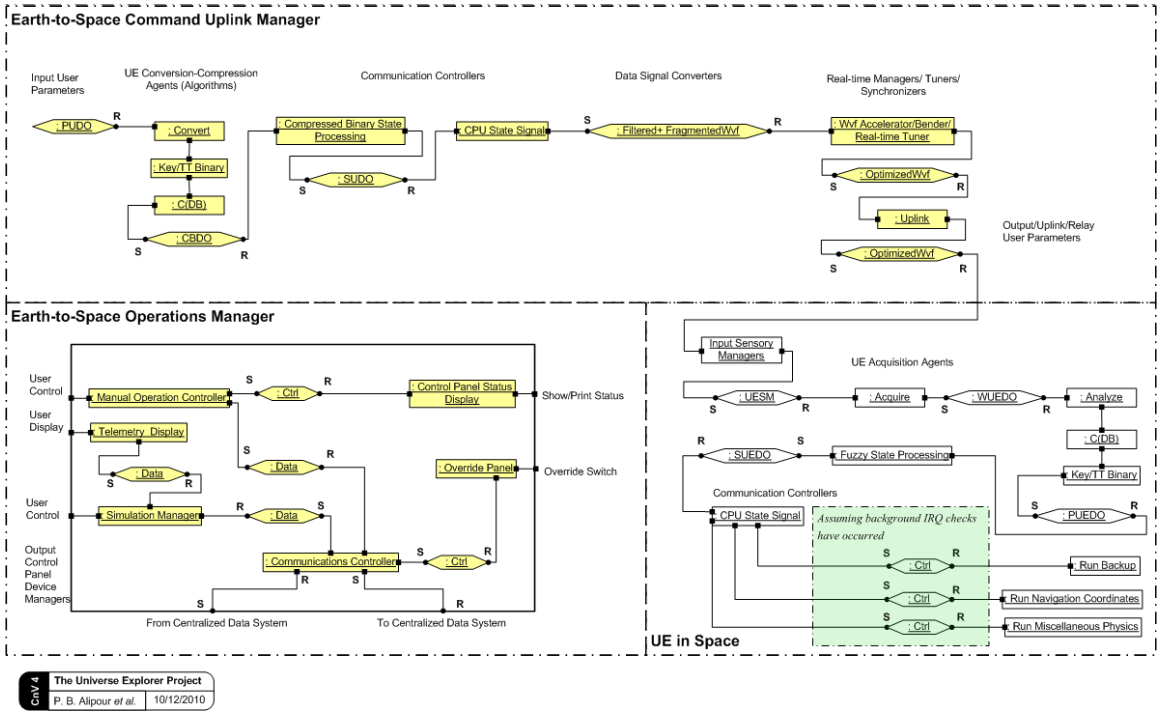

Fig. CnV5. Conceptual Configuration of the UE system from Earth to UE in Space

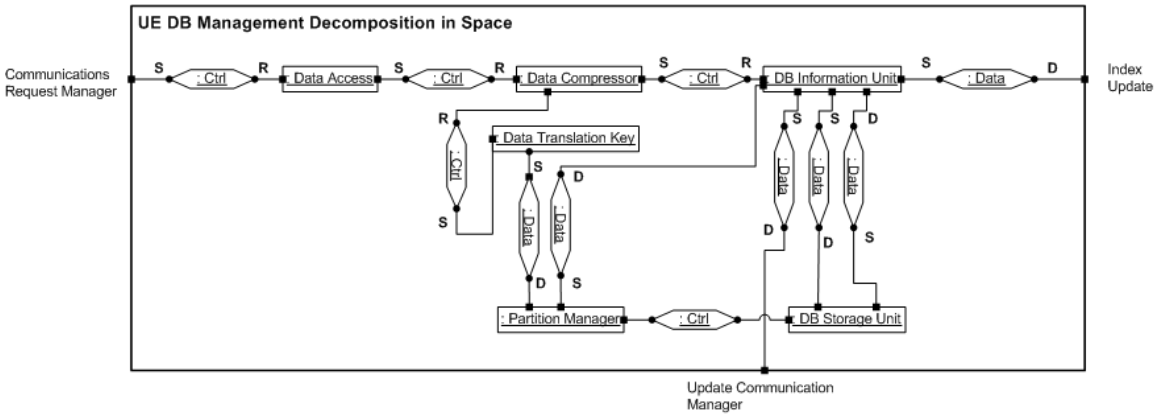

Fig. CnV6. Decomposition of the DB managers with lossless data compression technology in Space

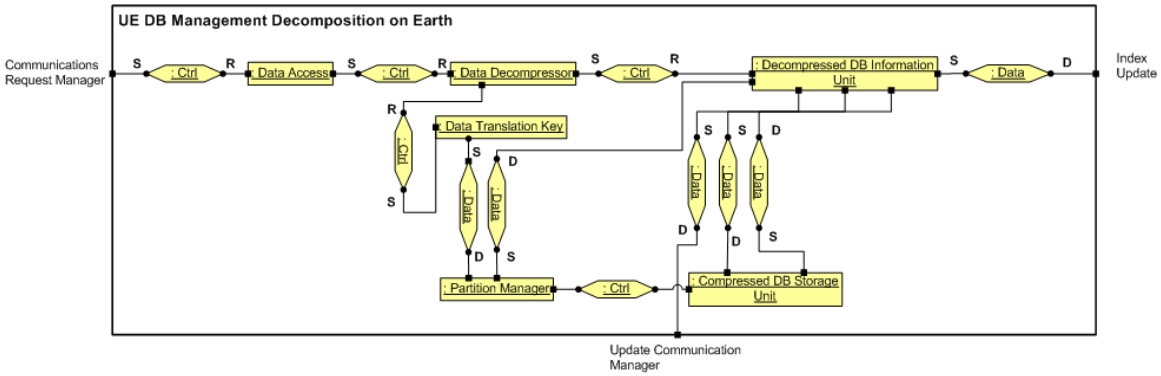

Fig. CnV7. Decomposition of the DB managers with lossless data decompression technology on Earth

In the last three figures, other UEDOConnectors are abstracted as Ctrl which is control and Data which is data, outlining where and to what extent functions apply between output ports and input ports as send (**S**) or receive (**R**) information, source (**S**) and destination (**D**) between entities specified back in § 3.1.1. SUDO in these figures, is the state of user data between entities. Fig. CnV5, shows the uplink process to UE in space from the station on Earth. The specifics of the connectors connecting one entity to another from the upper portion of the figure, sending information from base (station) to the lower-right of the figure (UE in Space), is recognizable in concept. The demarcated green zone over the three Ctrl's in the figure, are focused for controlling specific (one out of three) or a collection of hardware components (all three, dependably) by the processor based on the newly-arrived instructions from Earth. It could be a software update, or it could be a system-override problem like Bob's scenario to change course of the UE unit in Space to a safer zone, etc. (recall §2.1). The lower left portion of the same figure, in its the communications are formed between device managers and entities handling information updates (usually of evolved type) which could update their user for a worst-case scenario or even normal-cases through simulation, etc. as lately received from the satellite. This also involves entities that handle the overriding protocols in terms of registering reiterative values known to the system variables (or those security codes located at the centralized data system). Figs. CnV6 and V7, respectively represent the UE DB management decomposition in Space and on Earth. The main focus is either using compressed information for the UE system components at the software layer, or decompressed information at the station for humans or control panel user(s).

At the binary level, once data being compressed, the system does not have to know about the actual information and merely a representation of the binary code combination shall suffice using a readable key to the program. In other words, in concept an accessible translation key by the entity is enough to access relative compressed information to interpret rather than decipher the instructions onboard or even at the station. It is the other version that matters to a human cognitive sense when interpreting information at the control panel where data is graphically reconstructed (using e.g., decompression technique based on the pixel integer key, earlier given in § 2.1.1 discussion points) in Fig. CnV7. This, of course, involves the usability software quality characteristic in our architecture.

4. Module Architecture View

The standardized interpretation of pipes-and-filters architectural style is to address workable scenarios from our conceptual view in our architecture e.g., *acquire*, *analyze*, *compress* and *decompress* in a sequence, are deemed as filters (processing steps) for the UE, and are connected by pipes (channels) in a sequence to

output the expected product for the system. This leads to executable scenarios relative to conceptual and modular once revisited in the later sections of the report. Therefore, to satisfy such executable scenarios by hardware and software separate threads/co-routines in our system, we first must have an overview of the system from a module perspective.

In the module view, UE software is divided into application software and platform software. The application software of the UE system is organized into the following subsystems:

- **SignalAcquisition** is responsible for acquiring and preprocessing physics data from the Space environment.
- **SignalAnalysis** is responsible for deriving the parameters values and detecting Spatial conditions relative to the UE unit, as well as being of temperature type, pressure, etc. onboard and outside the unit.
- **DataCompress** is responsible for delivering compressed data to the system for further analysis when the processors need to derive specific logic states either to relay them to/from Earth otherwise, control or execute specific instructions on hardware components.
- **StateSignalProcessing** is responsible for pinpointing the logic state values and categorizing for specific component control after compression, which leads to communication.
- **Communication** is responsible for onboard communication signals or network in terms of runnable systems (miscellaneous physics), backup etc. inclusive of normal routines between processor and other components onboard. In parallel, UE communicates with other units located at the central station on Earth.
- At the station, the **control panel** is responsible for serving as the link between the user and the underlying functionality of the unit. It is responsible for the visual and audio presentation of information to the user. This could also be presented in form of a GUI or simulation pad in case of UE crises in Space.
- **User application** at the station is responsible for the high-level logic that implements the user requirements.
- **SatateControl** is responsible for coordinating major state transmissions such as startup, shutdown, standby and transfer both on Earth and Space.
- **SystemApplications** is responsible for power-supply management, diagnostic management, log management, and system resource utilization management both in Space and on Earth.

The **UserApplications** at the station, and the **SystemApplication** subsystems for both Space and Earth, decompose into device manager modules and entity modules. These are abstract modules that implement the device managers and entities in the conceptual view.

For example, there are entity modules in the module view for functionality such as data acquisition, compression, decompression and alarm detection on Earth, with simulation support, relative to space acquisition and compression functionalities in Space. Fig. MdV2, shows how entity modules are contained in UE's subsystems, upper image represents those onboard the satellite, and the lower image represents those at the station. These are some of the entity modules needed for the conceptual configuration of Figs. CnV4 and CnV5.

We did not require to include the 'decompress' module in our application software onboard the UE. The reason, as pointed out in § 1 (Fig. 3, Step description **iii**), is having an ASCII-based translation table (**TT**) algorithmic component as a *static key* for all compressed data, either being received from Earth/Space, or being sent to Earth, the UE system applications interpret compressed data via the static key which never changes during information update and application upgrades, as usable information onboard the satellite when necessary. Good examples from the usage point of the compression layer onboard the

UE, are further inducted in our Code View, §6 (see explanations on Fig.CV1 and the answer to Q.3 from the Code View evaluation, §6.1.3).

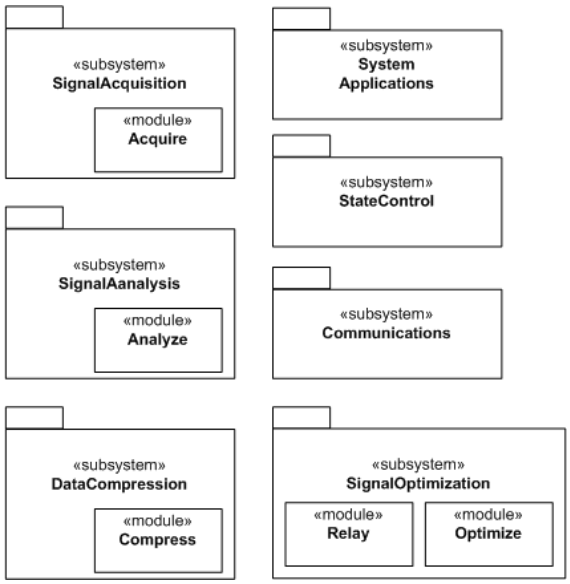

Fig. MdV1. UE subsystems in the application software

Fig. MdV2. UE subsystems in the application software at the station on Earth

An EntityModule is decomposed into an EntityControl, EntityData, and multiple EntityFunctions– one for each UEDO to which it subscribes (Fig. MdV3). It has an interface for each of the UEDO's it publishes or to which it subscribes. When an entity receives an UEDO, the EntityControl initiates the corresponding EntityFunctions for processing the UEDO. When publishing an UEDO, the EntityFunction returns control to the EntityControl. EntityFunctions share a common EntityDataSpace. Services provided by the platform software handle the publication and subscription requests, so that entity modules do not interact directly with each other.

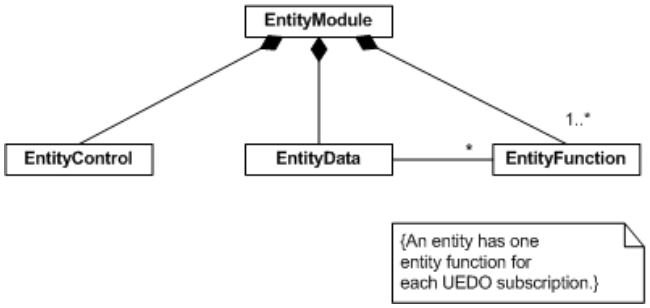

Fig. MdV3. Meta-model for Entity module structure

4.1.1. Decomposition of the Platform Software

Data managers

The platform software provides services for communication between entities, communication with devices, and other general system services. Its design is the result of applying the following three strategies:

1. *Use reentrant repository for data sharing* (issue Easy Addition of Features).
2. *Use central data manager and repository to exchange information* (issue Easy Addition of Features).
3. *Encapsulate the operating system and communication mechanisms* (issue Changes in Software Technology).

The platform software contains the following subsystems:

- DataManagement provides an interface for entity modules to publish and subscribe to data. Entity modules share information via UEDO's provided by this subsystem. It is responsible for managing the centralized repository in which the UEDO's reside and for providing services, for data publishing and registration, data compression, distribution, and access to local and remote data between Earth and Space systems (internal/external).
- DeviceManagement, defines a standard interfacing to product-line devices. The standard is based on **Assembly** to enhance portability. It is responsible for providing a stable, standardized device environment for the rest of the system. It protects the software from the hardware details, allowing applications and other subsystems to be ported to other hardware environments.

- SystemServices, provides an interface to the UE host platform. System services include compressed file handling, inter-process communication (IPC), logging interfaces, operating system support, system resource protection and time services. The additional system services on Earth would be to include system console interface at the control panel as well.

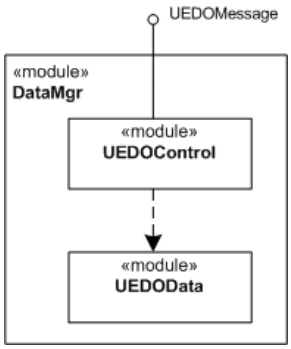

Fig. MdV4. Data manager

Entity modules access UEDO's through the DataManagement subsystem of the platform software. Instead of peer-to-peer communication between entities in the conceptual view, the Entity modules communicate via data manager in the DataManagement subsystem. The decomposition of the data manager shown in Fig. MdV4. The data repository is encapsulated in UEDOData module. The UEDOControl module provides the data manager functionality and access via an interface that adheres to the UEDOMessage protocol defined in Fig. CnV3.

The strategy, *Do not allow applications to share global memory directly* (issue Quality Assurance) applies here. The strategy, *Do not allow entities to make blocking calls* (issue High Performance) also applies here. In-line reads of data are not allowed unless an entity is the owner of the data. When an entity issues a data read or registration request to the data manager, the data request reply comes back to the entity in a separate message. This allows applications to be written without knowing whether data is coming from a local data repository, from a repository on a different processor, or from across the UE communication peers.

Having this additional infrastructure in the platform software meant that all software engineers were trained and could communicate using a common framework. For example, more than 95% of inter-entity communication uses common data distribution techniques and common event notification techniques implemented by the data manager [1]. Because there is only one way for entities to communicate, it is easy to explain and understand. This additional infrastructure eliminated reentry issues for entity modules due to global data being not shared between entities.

The platform software design ensures that moving to a faster processor affects the software only at lowest level of interface with the host platform. The software design makes it relatively easy to distribute the application software onto multiple processors.

4.1.2. Layering Structure

Software -Hardware Layers

The layering structure is used to implement the global analysis strategies for providing abstract interfaces and isolating dependencies. This structure, shown in MdV5, is orthogonal to the subsystem decomposition just described. Although all the modules in the platform software are assigned to the Platform layer, modules in the application software are spread over multiple layers. The module layering had part in motivating our UE model construction based on similar approach made in client-server models, since signal from space is in fact client.

Within the Applications layer, entity modules are further separated into policy makers and procedural application, to separate the functionality for product-specific requirements from more generic algorithms that are used across the product line.

The subsystems for applications software in UE fit into the layering structure in the following way.

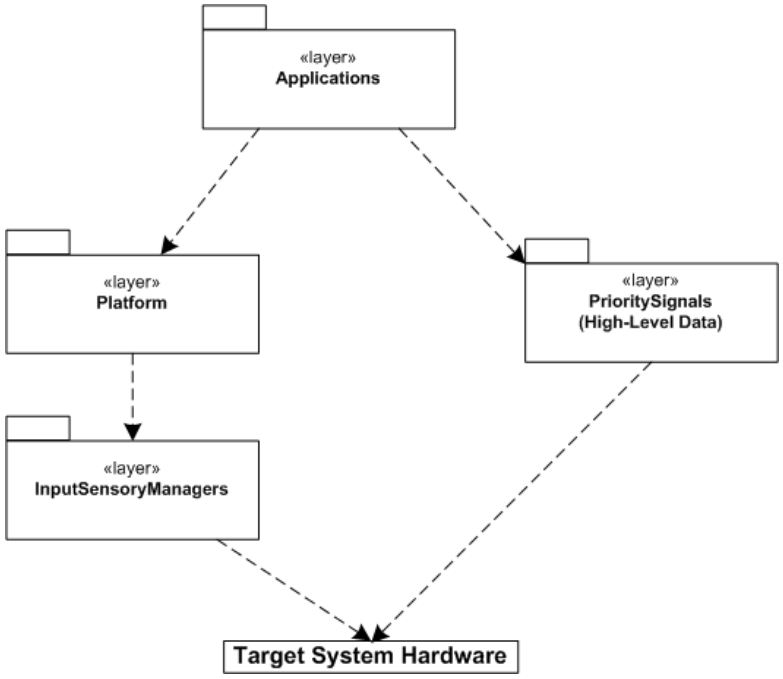

Fig. MdV5. Layering structure in UE

Entity and agent modules are located in the Applications layer. Device manager modules are located in the InputSensoryManagers layer.

The SignalAcquisition subsystem is the interface to the hardware devices that monitor the events of the internal/external UE system (see Fig. 6 right image, specifying the internal and external systems). It consists of device manager and agent modules that convert the incoming signals into UEDO's. The device manager modules are located in the InputSensoryManagers layer, and the agent modules are located in the Applications layer.

The SignalAnalysis subsystem is in the Applications layer. It contains two kinds of modules: procedural application and a policy maker. The procedure application analyzes the acquired signals. They contain generic algorithm used in product family that were developed independently from and prior to UE. An entity wrapper around each algorithm ensures that the resulting code follows the architecture model. The policy maker entity synthesizes the information from the algorithms in form of compressed information using a key which the latter subsystem (DataCompression) is too located in the applications layer, thereby reports it to the modules in the StateControl subsystem (at the station on Earth would be UserAppllications subsystem).

The StateControl subsystem, which is also in the Applications layer, is the chief policy maker. It provides top-level coordination for the operational states of the UE system (e.g., monitoring, standby and shutdown of parts or even the whole system)

The written or even ported code as an update to the UE system in space, was partitioned into device manager and agent modules to follows the architecture model.

4.1.3. Error Logging

The Building Blocks to Evolutionary Agent-Based Programming

Because error logging at a *binary level* is used throughout the system, it is important to *minimize the impact of any changes to code design* by creating a separate error logger module and guidelines for its use.

The error logger allows callers to log an entry containing the filename and line number were the error was detected. As many as nine 4-byte integers of information, including an error code, a severity level, the time of error, and a trace of addresses and arguments of the call can be logged, and provided to more intelligent or evolving algorithms. This gives the context for self-debugging and fits within the resource constraints of the embedded systems, in accordance with the strategy *create RAM-based error logging* (issue Quality Assurance).

One of the usage guidelines is that, services do not produce fatal errors if an input argument is invalid. Instead, the service passes an error back to the caller and the caller must take appropriate action (exception handling). An error log can be examined locally via the trained algorithm (*evolved*) over the communication lines onboard and even could be examined from Earth at the station. The error logging could also represent fatal hardware failure which switches to the backup system, and the system call complies with new assigned policies made by the intelligent algorithm for future potential encounters, in space or aboard the satellite (internal components).

5. Execution Architecture View

From an execution viewpoint, we have considered the usage of compression techniques to our I/O information between the DB system and processors. By considering equipartitioned data/fragments,[9] as *equal-sized chunks*, feeding it to the FBAR LDC algorithm with its static key (always static in size), generates spatial fixed ratios as well as execution. This means, the temporal property is predictable under almost all circumstances between central processors and storage.

The strategy is to preemptively plan for static memory (space) access of specific data, before the evaluation process of the architecture, resulting reliably-internal executions onboard the satellite, as well as the communication units at the station on Earth relative to their centralized data system. This is in contrast to regular data compressors which heavily rely on Shannon entropy, thus, generating unpredictable spatial and temporal scenarios per se. Furthermore, our communication units at the station shall deal with these equal sized packets, and no matter the time length of execution, scalable down to real-time scenarios based on the hypercube technology installed as many-core CPUs recognizable in the supercomputer and server-mainframe of the system.

---

[9] The notion of having "*equipartitioned fragments*" is to address the bottleneck problem on the overall stored data in terms of *fragments* which decreases performance. However, equipartitioned fragments have the opposite meaning, giving *high performance* ability for the processors to process the partitioned version against/compared to its unpartitioned version which is purely fragmented (bad performance).

The external execution from Space-to-Earth and Earth-to-Space, on the other hand, bear the barrier point of light speed which has been clearly elaborated in § 2.1 on our worst-case scenario factors. This means, this latency is unavoidable when the Satellite reaches farther distances in space (recall Bob's worst-case scenario), whereby the only solution for this, is hypothetically including/assimilating PTDV-SHAM [22], as a time-varying memory component, sender-receiver onboard the satellite, which superposes the signal from source to destination.

So, the overall execution by average, is a number of finite time steps between communication units in Space and on Earth with respect to real-time performance, which should be a ratio of 1:*n* as rationalized in §1.2.2, of our evaluation.

The time cycles between CPU and its components, benefiting from the basic I/O instruction set software stored on an EPROM, must be as short as 2 time cycles. This is maintained in terms of signal_send and signal_recieve between the cores and components denoting interrupt service routines, later known as AWK and IRQ signals (initially inducted at the conceptual level, § 3.1.2.) relative to other forms of instructions for inter-process communications. Such communications could be initiated as CYC and R/W, which should be known to the system designer and programmer. Since the CPU technology uses as multi-core to many-core in its architecture, we expect a hypercube performance on any errors to be detected and corrected between the core data process dimensions (see also Murdocca & Heuring, §9.4 [50]). The number of dimensions *D*, to compute time, could be given by

$$\log_2 n = D \tag{4}$$

where *n* is the number of processors, or in this case, cores with cyclic instruction tasks to communicate with other hardware components. Thus, shortening the time cycles with more cores is based on more communication paths using the above equation, which delivers more routing points (or *n* connections meeting at each processor/vertex). Therefore, a 2 cycle/2*n*-core attaining real-time scenarios is a fact of practice on making our execution/time behavior as efficient as possible. In return, we also expect the very notion of communication delays onboard the satellite between analyzers and acquisition agents normalized and consistent by the many-core CPU solution.

Guidelines for designing tasks call for a task assignment that minimizes operating system context switching and those overhead, increasing system throughput and reducing processing latency has been already strategized and illustrated using e.g. soft-switch hypercube solutions and semaphores like in Fig. 4. Fig. ExV1, uses a UML Deployment Diagram to show the processors located in the base UE unit, and the communication paths between them.

The choice of operating system was guided by the strategy *Use the UE RTOS real-time operating system* (issue High Performance).

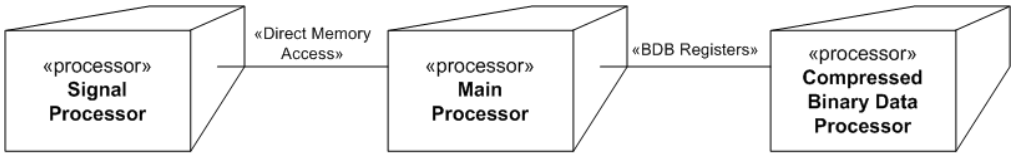

Fig. ExV1. Processors for UE's base unit in Space

Fig. ExV2 shows some of the platform elements provided by the RTOS operating in Space while having a *mirrored version* at the station on Earth. There is one common address space shared by a number of tasks and interrupt service routines. To create an instance of a task, addressing task allocation issues, the programmer assigns to it a message queue and a program counter. Entity modules are assigned to tasks, and device managers are assigned to interrupt service routines.

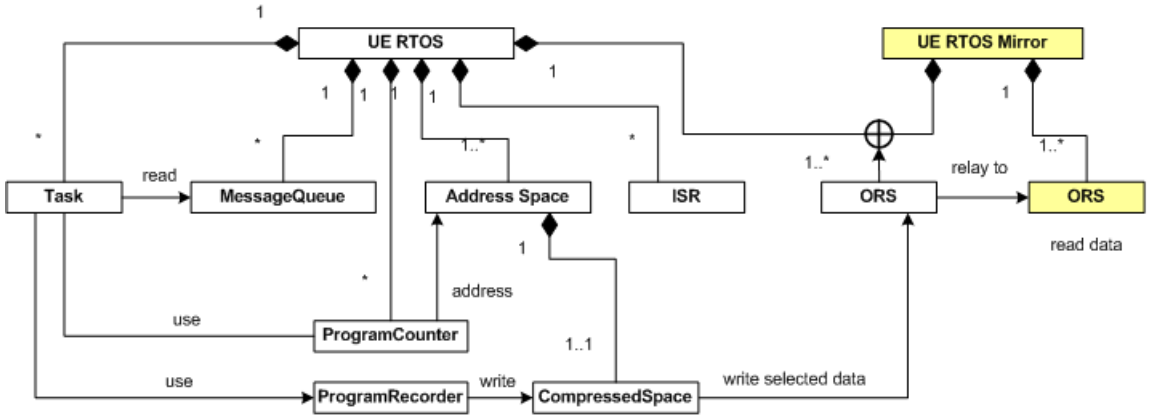

Fig. ExV2. UE RTOS platform elements used in UE. ISR = interrupt service routine; ORS = optimize and relay signaler.

5.1.1. Defining Runtime Entities

Task Assignment Meta-model and Execution Deadlines

The strategy *Create a task for each unique processing deadline* (issue High Performance) was used to assign entity modules to talks. Entity modules are organized into priority groups during resource budgeting. There is one priority group for each processing deadline found in the timing requirements of requirements specification. Then each priority group is assigned to a separate task.

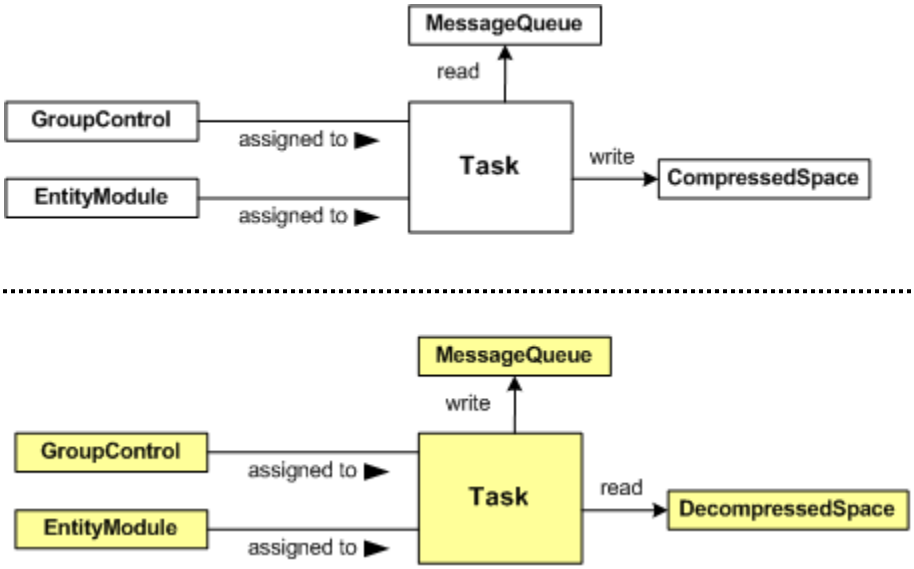

Fig. ExV3. Upper-image: Meta-model for UE task structure in Space; lower-image: meta-model for UE CCS task structure on Earth

UE tasks have the structure shown in Fig. ExV3, the upper image for the task onboard the craft and the lower for the task at the station on Earth. A priority group is implemented in the module view by a

GroupControl module. The GroupControl module contains the control logic software that routes the message to the appropriate entity (EntityModule). Messages contain the UEDO's to which an entity has subscribed. The entity module has its own control logic (EntityControl) that selects the appropriate entity function based on message type. When message processing is complete, the entity module returns control to the GroupControl module.

Because entity modules receive all inputs from the GroupControl module and do not make locking calls waiting for inputs, they can be moved among processors and tasks. The assignment of an entity module to a task is based strictly on timing deadlines. If a timing deadline changes, it is very easy to move an entity module from one task to another. A tool uses a binary description of the GroupControl module to generate a new *source file* for it. It is recompiled, and the software is re-linked.

The concept of priority groups made the definition of the tasks in the execution view straightforward. A task was created for each priority group. For instance, the requirements give the maximum delay for displaying waveforms and audio at the station, including alarm information, as seen in Table 3 or below.

Processing deadline	Maximum Delay	Delay Reduction by 2*n*-Core Inclusion with relative Adaptive Software Algorithms (Semaphores)
In Space		
Main signal state processing delay	0.8 s	800 msec.→≈0 s
Compressed state processing delay	0.4 s	400 msec.→≈0 s
Local waveform processing delay with compression	0.2 ~ 0.3 s	200~300 msec.→≈0 s
Relay pulse delay to Earth	1.1 s to ∞ s	∞→≈0 s
On Earth		
Local processing delay on Earth apart from simulation processing *****	0.4 s	400 msec.→≈0 s
Drift in blip/beep stream	50 msec.	50 msec. →≈0 s
Alarm annunciation delay	500 msec.	500 msec.→≈0 s
Relay pulse delay to Space	1 s to ∞ s	∞→≈0 s

The explanations on both Space and Earth of the maximum possible delays with subtractions are given in § 2.1.1, right after product factors.

In a couple of cases, special tasks needed to be added to meet deadline that arose due to the way the code was implemented, initially. An example of this is the 2*n*-core communication code or multithreads. Although, the logical deadline was 400 msec. the way the code was designed meant that the networking activity had to be completed within a few tens of milliseconds. The new task was created to meet this timing deadline. The end result was 27 unique deadlines giving rise to 27 tasks where 9 where executed concurrently. Thus, 27 – 9 = 18 tasks is the end result. The cost of adding tasks was a concern because each task has dedicated stack space and the system has reasonable amount of memory space. However, compressed shared space was considered to execute tasks concurrently without interfering specialized tasks for a specific set of modules. To keep the stack size under control, the genral rule for developers was that temporary buffers of k×100 kB maintaining the minimum communication levels intact with the system (see Step 3, Fig. 1), and thus the larger buffers should be allocated from the heap regardless of having a compressed reserved space to share tasks from one another.

5.1.2. Communication Paths

Main Processors and more on Deadlines

Fig. CnV4, in the Conceptual View, shows two types of connectors between entities. The UEDO connector is typical connector between entities, and its communication is as follows:

When an entity produced new UEDO data, it uses data manager's write managers. The data manager maintains a list of entities registered to receive updates of a particular UEDO. In addition to updating the UEDO data in the data repository, the data manager calls the IPC service in the SystemServices subsystem. This service maintains the mapping between an entity module and the signal collection send queue of the task in which it resides.

To meet the stringent, real-time requirements of acquiring and displaying the waveform data, an additional communication scheme was developed. This is represented in Fig. CnV4, as an optimized Wvf connector + filtered data onboard. The UE system must keep up with the hardware acquisition of the incoming data either in Space or on Earth, by accepting new data every 10 msec. and displaying it within 1 second, assuming the range of ∞ sec to reduce down to $\approx$ 0s or real-time, addressing the light barrier problem for far distances in Space compensated by the Wvf optimizer. Processing the data has a less stringent deadline. The algorithms for signal analysis relative to compression expect data every 400 msec. The overhead of writing more than $2n\times100$ waveform streams (proportional to the $k\times100$ kBps bandwidth factor) to the data repository to meet the 0.2 + 0.4 = 0.6 s deadline requires more than, thousands of calls per second, which could be handled by the *n*-core semaphore soft switch solution (Fig. 4).

The OptimizedWvf connector is implemented as a direct pathway between entities. During parallel processing, the filtered information (compressed) gives the optimizer (or even Wvf accelerator) a bypass in case of having an evolved agent-based algorithm (anEvolvedEntity) the opportunity via the second processor (focused on compressed data), to bypass the data managers as a mechanism for transferring waveform data or selected relays of repeated signals (repeated data). Thus each waveform is written to the data repository once every 400 msec., which saves significant CPU time before being relayed to Earth. To decouple the acquisition entity from the waveform relay, the entity's processing requirements, the waveform relay unit, tells the acquisition entity via its ABL, which filtered waveforms it wants to relay rather than wait for it to get processed.

5.1.3. Execution Configuration

Main Processors and more on Deadlines

Fig. ExV4, shows a representative set of tasks for UE's main processors. The interface to the signal processor is through direct memory access. Data queues provide the interface to SignalState processors via the UE host interface registers. Although all of the tasks have a message queue to simplify the figure, not all of this is shown.

As previously mentioned, entity modules that are defined in module view are assigned to a task-based on processing deadlines. Modules with similar processing deadlines are placed in the same task, and the task itself is assigned a deadline. The longest executing function in the entity must takes less time than the deadline of the task. Task priorities are based on the task deadline so that tasks would a shorter deadline have a higher priority.

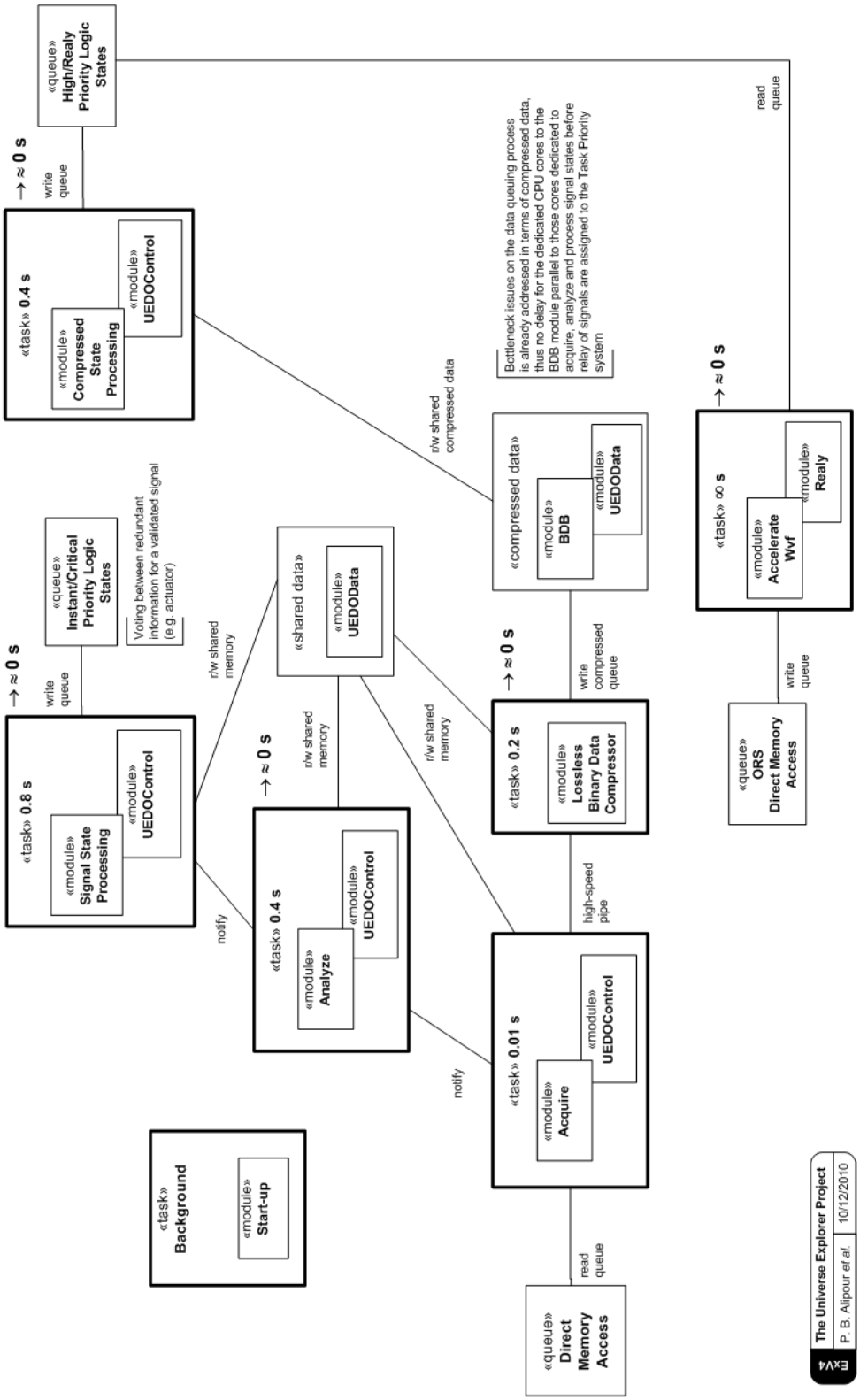

Fig. ExV4. Tasks on the main processors in UE system

The tasks range from the highest priority 10 msec. task, instant priority, and relay priority, down to the lowest priority background task. The background task is the result of the strategy *Implement watchdog mechanism* (issue High Availability), and it uses all spare CPU time to verify that the software image in terms of compressed binary is not corrupt.

The default connection between tasks is via the data manager. The data manager is divided into two modules: one for control, and one for UEDO data. The UEDOControl module is linked to each of the tasks, and it controls the access to the module containing the shared UEDO data and notifies the tasks that have subscribed to the data. An exception to the default connection is a direct message passing between the 10 msec. and 20 msec. compression tasks, which provide a high speed pipe for passing waveforms. The other platform software, device management and system services, is placed in binary libraries on the BDB partition or compressed space with a key, and is also linked statically to each of the tasks that require these services.

The entity concept could have been supported on these processors by making minor modifications to the data management and IPC services.

A key strategy for the execution view is that all tasks are created at startup; tasks are never dynamically created or deleted. A very different architecture with the different rules about bypassing vs. blocking would have emerged if each evolved entity, dynamically created and deleted its task on demand. This of course was not the case due to proper partitioning of data and handling it concurrently through a compressed shared data and memory space.

We further show a Code View architecture which introduces evolutionary programming to handle specialized task and bypass data management tasks based on a learning algorithm in our system.

6. Code Architecture View

The code view is imperative on issues related to *functionality*, *maintainability*, *efficiency* and *reliability* from the software quality evaluation model ISO/EIC 9126-1 [38] on the UE components. The components' data is processed, and thus controlled by the 2*n*-core CPU 02, parallel to incoming external data from either space or Earth by 2*n*-core CPU 01 and data acquisition subsystems' domain (recall Fig. 1, and the two CPUs evaluation made in § 1.2.2).

We have therefore further analyzed our UE system requirements to also include a code architecture satisfying the four quality attributes' set from the evaluation model mentioned above.

6.1.1. Code Development Process, Scenarios and Architecture:

Code, Choose, Build, and Deploy

The code view mainly addresses issues raised in the early sections of the report on the backup subsystems including controllability over hardware components on board the satellite (last paragraph). The time cycles relative to code execution, must be maintained in terms of signal_send and signal_recieve between the cores and components, shortly known as AWK and IRQ signals (as given at the conceptual level as well as execution view, §§ 3.1.2 and 5.)

The following pseudocode shows how such issues must be addressed real-time in parallel to the routine operations aboard UE. As we have overly-explicated that, in our requirements, we initially need at least two CPUs comprised of multi-core nanotechnology (§ 1.2.1) with their memories to be installed on-board the satellite.

The following pseudocodes distinguish their tasks from the main core of practice between components' managers (within the CPU 02 domain), acquisition units, and processing (CPU 01).

Pseudocode sample I:

```
If {C1, C2, C3, …, Cn} = True
Print "OK"        //status report on components C1 to Cn list
ElseIf Ci = False //a particular component from the list
Print "Ci is not OK"
Send backup signal to Sat for backup confirmation or AWK signals
Init backup components or Ci'
Repeat Ci procedure for Ci'
End If
```

The above pseudocode shows a list of UE hardware components on-board and runs real-time tests/checks inspecting failure (unhealthy or error) otherwise acknowledgment signals (healthy status using AWK). The test is in fact, merely a monitoring session or status of the components run by CPU02 as i.e. IRQs (interruption request signals) which in return converted to AWK signals or

$$\mathrm{IRQ}_i \rightarrow \mathrm{AWK}_i = 1.$$

Each IRQ comes from a particular component (shown as C_i from the list), and the CPU checks whether the cycle or IRQ is being sent on the expected time cycle, thus acknowledging the healthiness of that component (AWK = 1 stands for true). Otherwise, there is a problem concerning the component's status. Therefore,

$$\mathrm{IRQ}_i \rightarrow \mathrm{AWK}_i = 0 \rightarrow \mathrm{IRQ}_{\mathrm{CPU02}} \leftrightarrow \mathrm{IRQ}_i',$$

where the last two IRQs, (one broadcasted by the CPU to the backup component, and the other, the response from the same component), establish a communication path between the CPU and a backup component or alternative path to replace the faulty component with a new one e.g., activating the circuit-breaker in Fig. 1, §1.2.1. This is done by initializing (Init) the relevant procedure(s) on the faulty component C_i now replaced with a new one C_i'. From now on, CPU 02 checks the new component in terms of $\mathrm{IRQ}_i \rightarrow \mathrm{AWK}_i' = 1$.

In parallel, CPU 01 despite of such worst-case scenarios, is doing its job in terms of receiving incoming signals either from space or Earth as specified, back in Fig. 1. For example, suppose the developers on Earth develop a new code for UE applications upgrade purposes. They thus send the code to the satellite or uplink it. To confirm whether their code is reliable enough or compatible aboard the system, the program or the new algorithm is executed. The system tests and checks that everything is working according to plan. If e.g., incompatible for some OS reason, the current BIOS system aboard the satellite collectively allows the minimum restoration protocols to take place i.e., triggering CPU 01 to execute a cloned version of the previous version via memory, which restores all programs *by the cloned compressed database or partition* which benefits from the standards known in Blob computing [53] or binary DB which in this case represents compressed data (binary data stored as a single entity). This scenario is visible in the following pseudocode where it executes the newcomer code for all components handled by CPU 02 subsystem domain. If not working properly in terms of AWK and IRQs as a pretest condition, CPU 02 ignores the seriousness of this release and asks CPU 01 to carry out restoration protocols. CPU 02 ignores this release because it is a new arrival and according to the code header, must first pre-test a new version on a trial basis. If it turns out that everything is working fine, the system proceeds with the new version protocols, replacing the old protocols with the new one. The following pseudocode is too high-level, but shows how this scenario takes place under upgradability circumstances.

Pseudocode sample II:

```
Deliver new code to CPU02 via CPU01
Run new Code on {C1, C2, C3, …, Cn} //CPU02 is pre-testing
```

```
While CPU02 pre-tests {C1, C2, C3, …, Cn} Do
Old Code = New Code              //assign new values to the old variables chronologically
If New Code or Algorithm = False  //assignment becomes false in the pre-testing process
Execute restoration program
Else
Proceed with upgrade       //CPU02 is OK with post-testing
End If
End While
```

The directory structures in Figs. CV1 and CV2, are highly fundamental to the execution of the subroutines and thus their decision node complexity of the code [45] on synchronicity vs. asynchronous communication paths between the CPU 02 and other components (or subsystems). Fig. CV1 (or code view1) specifies the directories before being deployed on the UE spacecraft. This aids the programmers to know what they are doing during the project development cycle. The documentation or comments directory, [Doc/Comments], is where documentation outside the code and/or comments within the code are presented for the programmers during development as their guide or instructions manual. These programmers program the codes in ANSI C for memory efficient purposes (mostly hardcoded and avoid dynamic ways that require more processing to spare with), and use a set of coding conventions, most of which are standard. For example, to ensure the names of the UE hardware components, they are presented in an enumerated list (enum) starting with the name of the enum itself. If the enum type was C as component according to our pseudocode above, the main CPU component might be C_CPU, and the blower component might be C_Blower. However, on a defined efficient level for the compiler, such parts should be presented in terms of e.g., C01, C02, C03, …, depending on the level of the component's role or priority.
Fig. CV2, on the other hand, is the actual directories we require on-board the satellite to get executed which all data are represented in binary (Bin). The rationale to this selection shall be clarified shortly when the system self-tests, and executes data on all of its components, and thus establish communication as a *self-automated* approach in its architecture.

In Fig. CV1, the source code for the development team on Earth denotes the actual whole code of the hierarchy. Thus, the [CodeRoot] retains all directories which need to test the hardware components on-board the UE satellite. This is done via the [Test] directory containing test codes. On the other hand, the [EntityParallelCode] represents all multithreading or parallel coding procedures needed on an application level to control the hardware components via [SubsystemsCodeCPU02] *parallel to* [SubsystemsCodeCPU01] (parallel time execution paths). The latter, "SubsystemsCodeCPU01", however, processes data on a 'compressed DB level' and does not care about other hardware components onboard. In fact, its specific task is to process incoming data from space, and outgoing data to Earth, and vice versa. This means that, CPU02 is dedicated to certain instructions satisfying functions of other keys of the UE system, in terms of monitoring other components while controlling them (CPU 02's job). In the meanwhile, CPU01, according to specification, is here to process events after data acquisition techniques on the incoming signals from the space environment whilst giving out signals via the compressed database (DB).

So, in return, the build-on compilation process on Earth, or [build-on Earth] directory, generates a file, which is presentable to the developers' manager, representing that the UE is functional on all terms of hardware and software, and thus transmission protocols indeed.

In turn, once the devolvement team finishes the project on all levels of code implementation, they test ([Test] directory) all of the components and thus expect the right results while the UE system learns if any errors occur. This is handled by the [ABL Control] directory which stands for agent-based learning algorithm (typically, a *learner*). It is almost blank in terms of code-depth or layers (highly primitive) at first (a very short algorithm per C_x variable or component management), and has a set of single nodes.

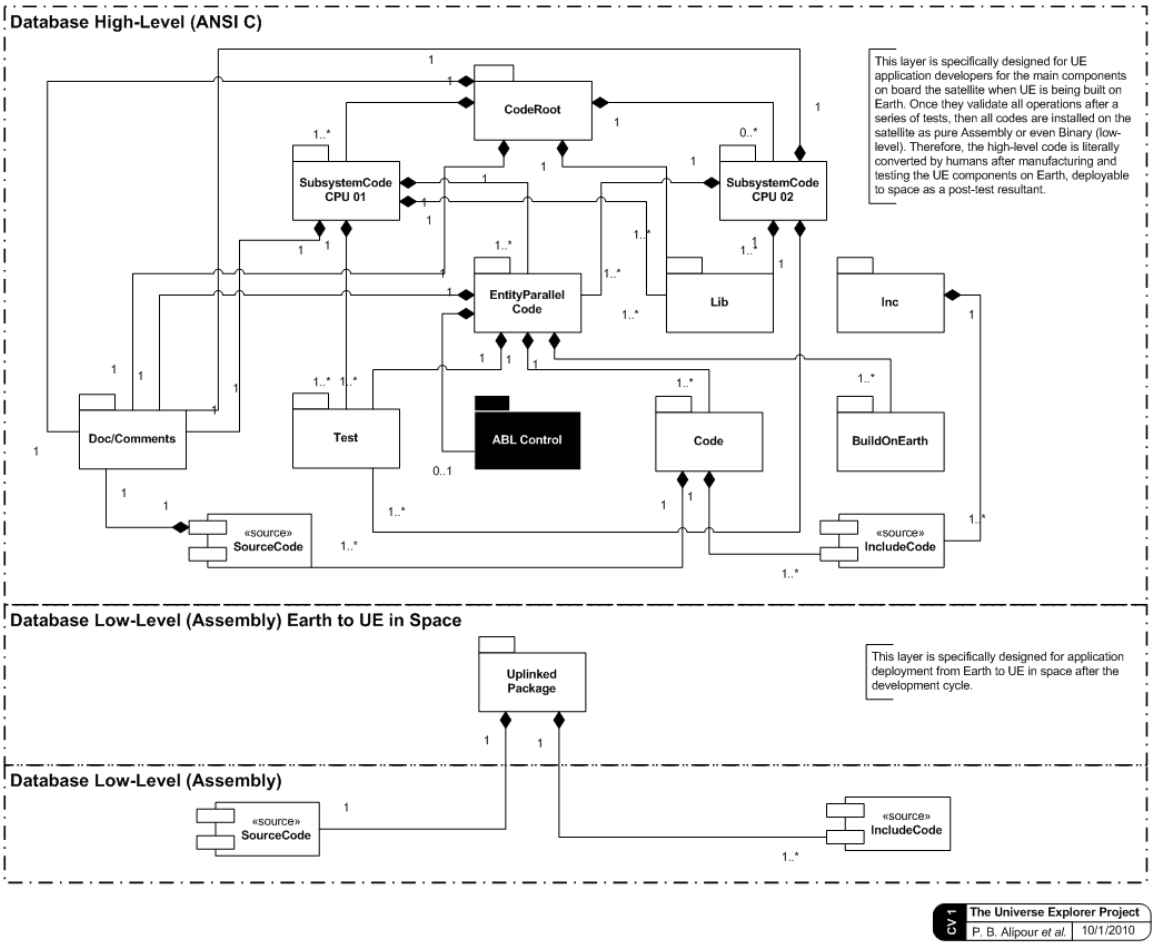

Fig. CV1. UE system data process and control directory structure, before deployment to space, on Earth

These single nodes should learn from the previous mistakes (errors, etc.) and create new nodes for reiterated negative encounters as the system evolves in terms of upgradability once the ABL monitors in parallel all ongoing events of other codes (from other directories) via CPU 02. CPU 02 also monitors CPU 01 as one of its primary tasks relative to other hardware components onboard the satellite. The ABL eventually evolves throughout the course of UE mission and would become a network of nodes, intelligent enough to even get upgraded unless it becomes a problem e.g., a virus or an invalid algorithm in the mission. Of course, downloading this virus in aim of studying the problem is vital for future application upgrades which must be done on a securely confined database (one of the server partitions) at the station on Earth (see Fig. 3).

Hence, the overriding protocols take place, and the developers on Earth develop a new ABL code as they uplink to the satellite. This new package, denoted with the [UplinkedPackage] sends all vital information or upgrades to rest the UE system (like the rebooting procedure on a PC), which gives the system a fresh start of its operations. (That is why, the multiplicity is 0…* association, where 0 in this case is a blank document or the erased version ready for an Append, Write, Close, … commands from the newly-arrived package.) This package is transmitted in pure binary, and the processors must solely deal with binary values as the most efficient standard to their operations. Similarly, deriving from the Pseudocode sample II scenario, we have a 0…* association on [SubsystemCodeCPU02] with the [CodeRoot], due to being of a source code nature, whereas during uplink, it could be erased and thus

replicated from scratch. The previous/current version is stored as a backup for “restoration purposes” as specified in the scenario.

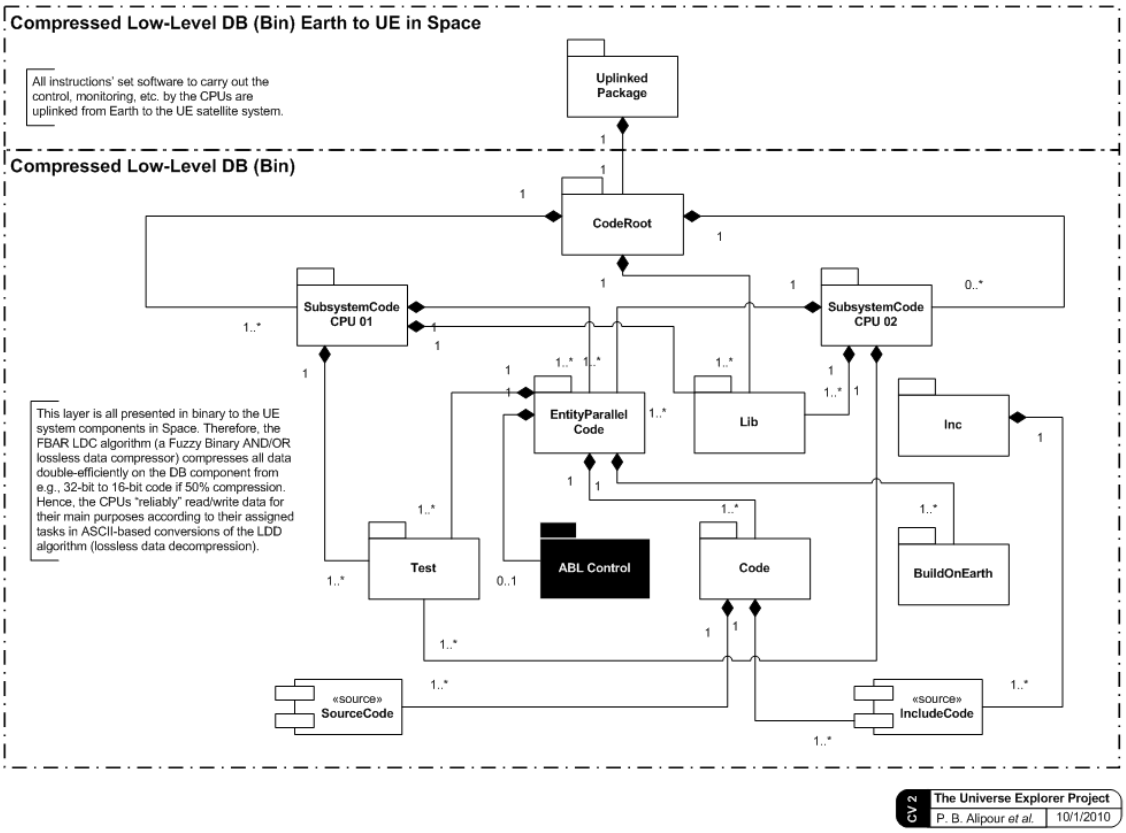

Fig. CV2. UE system data process and control directory structure, *after* deployment to space/after launch

Therefore, for our requirements’ engineering, the programming language on updates must be a machine language, highly-deployable to the UE units from Earth, and efficient, in terms of hardware-software component management issues by either CPU.

Bear in mind, in both figures, we have avoided too many connections drawn from the lower directories to their higher ones, relative to the component level (source code), by simply saying that the connections are inherited from an upper to their lower, in this directory structure.

6.1.2. Development Strategy with UE ABL:

Choose, Learn from Mistakes, Build, and Reconfigure the Code

The developers develop the code and maintain low-level programming e.g., Assembly, by their manual code conversion as they type (Fig. CV1), since the technology mainly deals with LDC techniques for its data transmissions between space and Earth.

Therefore, e.g., if the enumerative mnemonic type is the MOV instruction in x86 Assembly, data unit displacement in terms of *load*, *copy* or *store* from one register to another, must also be presented in binary without disrupting other communicators onboard, especially CPU 01. In turn, the LDC while having LDD

outputs via a key (FBAR **TT**) installed, [10] must also not interrupt such needy operations between registers and reference point to carry out a certain operation, like real-time test on components which concern the monitoring aspect of CPU 02 domain. The following could be a good example on a binary packet instruction level change from 16-bit to 32-bit (from [42]), where double-efficient LDCs relative to such changes in compacting data would still remain intact on this operation. Since such lossless compressors deal with 8-by-8, 16-by-16 bit, 32-by-32 bit… binary-to-any data type and vice versa conversions, the packets in their data conversions or encoding techniques e.g., if 50% LDC, thus the 32-bit is 16-bit at the compression phase, and thereby at the decompression phase, gives out 32-bits expectably (see also abstract [6]).

```
MOVZX EAX, BX
```

This code moves the higher-byte of the 16-bit register BX into the 32-bit register EAX.

So, this requirement, once implemented on the instruction set compression-level, also fulfills the *compatibility* issue between hardware and software technologies during UE software upgrades as uplinked by humans via the CCS system from Earth.

These programming assignments are mainly focused on the DB transactions' site maintaining communications protocol when an uplinked source code is installed on the system, while incoming data from the space environment is processed in parallel.

From our execution view, we can understand that why it is imperative to have our OS as low-level as possible, which also provides us error logs on the database at a macro scale, and thus an agent-based system (ABM), as the ABL onboard the satellite, aims to learn from those errors which lead to efficient and reliable monitoring of all events between components relative to space environment.

The UE ABL which evolves from the UE system's previous mistakes from a stupid or blank state of minimal node evolution to a maximal node evolution, is then classed as being highly-weighted in terms of AI (artificial intelligence). For example,

Pseudocode sample III:

```
Algorithm A START
If x > n   //x is some sensory variable
Report error
Else
Resume
End If
HALT
```

then,

```
UE ABL START   //this is the UE ABL algorithm on x
If Algorithm A reports an error
Monitor events by other algorithms or CPUs
Record error detection
Record error correction
Create Algorithm B as a new node
End If
HALT
```

[10] This lossless data compressor (LDC) with fixed compression ratios (preventing any information probability or randomness on time and data), with its universal translation table (**TT**) was earlier introduced in **iii**, §1.2.1, or see [6].

where,

```
Algorithm B START   //this is the new node generated by UE ABL
If x > n
End or erase Algorithm A
Do correction       //execute the shortest path without error report and detection
End If
HALT
```

Thus, the generated new node by the UE ABL is more efficient than the classical 'Algorithm A' in terms of executables and cyclomatic complexity [45, 46], which is self-learned or experienced from the previous mistake(s), and instead of reporting, uses the shortest path possible to do correction via relevant kernel components (e.g., CPU02). Just like any wise human who does not want to repeat his/her own previous mistake(s) iteratively (like learning the major lesson from your own past history), the node eliminated the dependency on the 'Else' decision and executable points from the initial 'if statement'.

6.1.3. UE ABL Evaluation

Is it a Blackboard Architectural Approach? - What does it learn? - Is it a Risk?

1- *What exactly does the UE ABL learn onboard the satellite?*

Learning scenario initiatives: The unpredictability in the space environment becomes predictable in the long run by an evolved ABL control network, when e.g., UE experiences Jupiter's intense gravitational field that have affected too much some of the body parts, next time, when revisiting this planet or some similar planetary state out in space, the UE knows how to avoid getting too close to the planet's system. Thus, gathering data from a safer point is done preemptively. Another example is the Bob's scenario in §2.1, where UE went off scale on its sensory calculations regarding distance, etc. due to a solar burst. Next time, the ABL has got the sensitive information on this unique instance as a combination of previous events detected and corrected from Earth to UE. So, even if a similar burst occurs, Bob doesn't have to worry and uplink new coordinates to UE, in fact, UE resumes its course in the right direction!

Evolutionary programming: The UE ABL nodal creations (resulting in a learning network topology) leads to *evolutionary programming* for managing both CPUs' instruction set plus their communication paths for their interacting components. From the evolutionary viewpoint, the software packages used today are designed for *serial von-Neumann computer architectures*, which is a downgrade to our initial design representation made in § 1, supporting parallel processing indeed.

This limits the speed and scalability of these systems. A recent development is the use of *data-parallel algorithms on Graphics Processing Units* (GPUs) for ABM simulation [39, 40], and [41]. The extreme memory bandwidth combined with the sheer number crunching power of multi-processor GPUs has enabled simulation of millions of agents at tens of frames per second.

Thus, a micro scale adaptation of such algorithms to our UE system by the development team, and thus hardware strategists could maintain stability in the UE's 50-year mission, and is by teaching the units how to adapt under complex or even unpredictable circumstances, thus providing us information on a very efficient scale. This, as a result, entails Verification and Validation issues on those newly-upgradable components that are ought to be employed onboard the satellite with a main impact on the execution view.

2- *Is UE ABL based on a blackboard architectural approach?*

Our choice and motivation: The current ABL sample is definitely a relatively-growing node behavioral system reflecting its growing *neural network* design which interestingly, sometimes abstracts the design. For instance, based on its cleaver replacements or removal of certain software components (via Algorithm B from Pseudocode sample III) when necessary, the *blackboard architectural style* is evident and even if a component fails, the ABL will not fail, typical to a 'pipes and filters' architectural style. Furthermore, the varying performance issue from the blackboard style is less experienced aboard the satellite, since the supportive technology is 2*n*-core processors, however, the notion of *low reliability* in terms of *non-deterministic execution* or difficult to debug [49], must also be self-embedded within the ABL program. It is aboard this satellite that of course *low security* is in view for monitoring all removable/replaceable components relative to their main processors. This, however, could not be changed due to Earth base critical-related scenarios where overriding procedures to reprogram, its removal and replacement of the ABL component with other versions when affected (for some unknown reason), is crucial in our design. Therefore, the operators on Earth will have no problems in controlling the satellite system under critical circumstances.

3- *Isn't it risky to always have the ABL onboard when it learns more and more from the system as well as the space environment?*

This learning system should not be a risk, since the satellite should know enough via the ABL algorithm when sent to space. The database on the satellite will never be filled, since the information will be sent to the Earth base/station and stored there through proper relay of signals using tuners/wvf optimizers or compensators installed aboard the satellite

4- *Will the UE database onboard ever gets filled when the ABL grows?*

As we have on many occasions explicated that it would be bad system to have a compressed database filled. The system not only relays most of the raw data to earth, it will also only collate and store vital information relevant to the system after pre-processing on a temporary basis for its learning algorithm.

Furthermore, we have considered the fixed compression ratio standard via FBAR [6] to function accordingly (see early sections), and based on this compressed data, only having a *static key* (a **TT** file), all data is dealt with as the right information during pre-process and post-processing reads and instruction set executions.

The ABL is here to always avoid and tackle the risks coming from space as well as the components onboard in case failure. The ABL is not here to pose a threat to its system or to be a risk!

In either case, when this very component becomes a threat, the Earth station has got its overriding protocols intact with the whole system to be executed, as exemplified for the worst-case scenario e.g., Bob's protocols in action (recall §2.1).

7. Summary

The main foci of our architecture were on maintaining all-time communications between UE in space and Earth while UE sustains its structural integrity in space i.e. its main hardware via software managers. Because of performance requirements, *shortest*, *lengthiest* against *real-time data-send* and *receive*, there were evident constraints in Space, such as the problem of *external latency issues* when the UE system gets farther from Earth into space, the speed light barrier overcomes the communication systems in terms of

minutes, hours, or in its last portions of the journey, days and months. Although, we have shown that the many-core CPU solution with the right software, tackles problematic entity model and entity-to-task assignments onboard the satellite, the implementation solution on assimilating signal accelerators, tuners and compensators, etc., for compensating external latency issues, was a pre-emptive strategy to contemplate. Therefore, hypothetical technological factor like, strategizing the usage of the PTVD-DHAM component was relevant to this problem.

On the software part, however, the evolutionary programming entities is of importance to the hardware components on board, such as the backup system, prior to the controllers, communicators and processors, which should evolve in 50 years relative to satellite's adaptability to the harsh physics of space. The code evolution is based on responsive ways to "do correction" rather than "first detect, then correct" scenarios from the UE's previous physical encounters in space, like temperature, pressure, etc. To this account, we further used the read/write functions onboard the satellite, not only for the learning algorithm, also for the BDB and other memory components like the EPROM which is programmable for a new set of instructions to be carried out by the CPU relative to Space environment.

The fixed space-time LDC ratio solution, on the data compression technology, is relevant according to our technological factors on communication between Space and Earth, addressing issues related to high-performance, integrity and efficiency. Moreover, it is essential to have a BDB (binary database) compressed with fixed compression ratios rather than random compression ratios. If random, this affects our execution viewpoint i.e. random time behavior, but fixed ratios generate fixed time behavior as well as memory space occupation for the loosely coupled entities onboard the satellite. Therefore the 2*n*-core CPU solution from a runtime perspective for maintaining real-time performance by corresponding to the asynchronous communication between entities with a ratio of 1/*n* seconds (descending to values close to 0) is hereby reiterated.

Since the satellite whole system in space is not retrievable with affordable strategies, we thus had to also challenge current hardware technologies with those being tested under laboratorial conditions or latest products evaluated by the space industrialist.

In summary, for all the above issues, we were drove to use of these concepts:

- **Entities:** must of the system is composed of loosely coupled entities. Entities do not communicate directly with one another; all communication occurs via connectors. Inside the entity, the control is separated from the processing, and it is standard across all entities. Entities are non-blocking, enabling entities to be moved to different tasks.
- **Connectors:** UE's connectors use a data manager and a repository to support data sharing among components such as entities. The connector protocol is published/subscribed, with asynchronous responses to data requests. An optimization was added that aggregates messages to/or from the data repository. This lowers the IPC message rate by sending multiple data repository updates to an entity in one message.
- **Tasks:** the assignment of entities to tasks is based on the similarity of their deadlines rather than the similarity of their functionality. Like entity control, task control is standard across all tasks.
- **Mediators:** interaction between subsystems that use different interaction mechanisms is done through mediators which could evolve during the 50 year UE mission. Mediators provide uniform data access to both producers and consumers, hide the location of the data, and decouple both sides from each other. Agents and evolved agents are an example of mediators, mediating between software entities and device managers that use different protocols and different data formats.

- **Layers:** standardized API's are used to isolate dependencies among layers. UE uses interface binary libraries isolating the dependencies on the hardware, operating system, and device managers.

With these concepts as the basis for the architecture, UE was successfully built. These concepts were developed without deliberate use of design patterns or component/connector architecture concepts, which were not generally known at the time of the architectural design. Similar concepts were invented, but it would have been more tangible to have used existent technology explaining the efficiency aspects of our UE system.

References

[1] C. Hofmeiester, R. Nord and D. Soni, *Applied Software Architecture*, Chaps. 4-6, and 8-10, 2000.

[2] Mayo-Wells, "The Origins of Space Telemetry," *Technology and Culture*, 1963.

[3] G. Brown, "How Satellites Work" at: http://www.howstuffworks.com/satellite.htm/printable Accessed Sept. 2010.

[4] NASA, *Statement of D. S. Goldin (Administrator) National Aeronautics and Space Administration (NASA) before the Subcommittee on Science, Technology, and Space Committee on Commerce, Science, and Transportation United States Senate*, Sep. 23, 1998; or see http://www.howstuffworks.com/framed.htm?parent=satellite.htm&url=http://www.hq.nasa.gov/office/legaff/9-23gold.html Accessed Sept. 2010.

[5] Satellite Dictionary: http://www.satellites.spacesim.org/english/glossary/sz.html, Accessed Sep 2010.

[6] P. B. Alipour and M. Ali, "An Introduction and Evaluation of a Fuzzy Binary AND/OR Compressor," an A/1st-class M.Sc. Thesis, MSE-2010-21, Blekinge Inst. of Tech., Sweden, May 2010.

[7] C. E. Shannon, "Theory of Data Compression," *Redirected from EFF: Electronic Foundation Frontier group*, http://www.datacompression.com/index.shtml. 2000.

[8] International Union of Pure and Applied Chemistry. "Photodiode". *Compendium of Chemical Terminology* Internet edition. Retrieved on 16 Sep, 2010.

[9] R. D. Antonov and A. T. Johnson, "Subband Population in a Single-Wall Carbon Nanotube Diode," in *Phys. Rev. Lett.* 83, *APS Phys. J.*, pp. 3274–3276, 1999.

[10] R. Yan, W. Liang, R.Fan and P. Yang, "Nanofluidic Diodes Based on Nanotube Heterojunctions," *Nano Lett.*, **9** (11), *American Chemical Society*, pp. 3820–3825, 2009.

[11] "Beyond Batteries: Storing Power in a Sheet of Paper". *Eurekalert.org.* August 13, 2007. http://www.eurekalert.org/pub_releases/2007-08/rpi-bbs080907.php. Retrieved 2008-09-15

[12] E. Flahaut, R. Bacsa, A. Peigney, C. Laurent. "Gram-Scale CCVD Synthesis of Double-Walled Carbon Nanotubes". *Chemical Communications* **12** (12): 1442–1443, 2003. doi:10.1039/b301514a

[13] Y. Fan, X. Zhong, J. Liu, T. Wang, Y. Zhang, Z. Cheng, "A Study of Effects of Coolants on Heat Transfer Capability of On-chip Cooling with CNT Micro-fin Architectures by Using CFD Simulation," *IEEE proceedings of HDP*, 2007.

[14] Y. Fan, X. Zhong, J. Liu, T. Wang, Y. Zhang, Z. Cheng, "Computational fluid dynamics for effects of coolants on on-chip cooling capability with carbon nanotube micro-fin architectures," *Springer tech. paper*, **15**(3), pp. 375-381, 2009. DOI: 10.1007/s00542-008-0742-9

[15] Preventing heat escape through insulation called "aerogel", *NASA CPL*

[16] M. L.; Nuckols, . C. Chao J and Swiergosz M. J. (2005). "Manned Evaluation of a Prototype Composite Cold Water Diving Garment Using Liquids and Superinsulation Aerogel Materials". *United States Navy Experimental Diving Unit Technical Report* NEDU-05-02. http://archive.rubicon-foundation.org/3487. Retrieved 2008-04-21.

[17] M. Bryning, D. Milkie, M. Islam, L. Hough, J. Kikkawa, and A. Yodh, Carbon Nanotube Aerogels. Advanced Materials, **19**: 661–664, 2007. doi: 10.1002/adma.200601748

[18] M. Berger, *Nanowerk Nanotechnology Spotlight posts* at: http://www.nanowerk.com/spotlight/spotid=1619.php Accessed Sep 2010.

[19] Olexa P. Bilaniuk; E. C. George Sudarshan, (May 1969). "Particles beyond the Light Barrier". *Physics Today* **22** (5): 43–51. doi:10.1063/1.3035574.

[20] Bilaniuk, Olexa-Myron P.; Deshpande, Vijay K.; Sudarshan, E. C. George (1962). "Meta Relativity". *American Journal of Physics* **30**: 718ff. doi:10.1119/1.1941773.

[21] G. Feinberg, "Possibility of Faster-Than-Light Particles". Physical Review 159: 1089–1105, 1967. doi:10.1103/PhysRev.159.1089.
[22] P. B. Alipour, 'Logic, Design and Organization of Prallel Time Varying Data and Time Super-helical Memory As; PTVD-SHAM', article id. arXiv: 0707.1151, *CompSci. Ar. ArXiv.org.*, pp. 1-33, 2007.
[23] P. B. Alipour, 'Theoretical Engineering and Satellite Comlink of a PTVD-SHAM System', article id. arXiv:0710.0244, *Comp. Eng., Finance, and Sci. (cs.CE); Hardware Architecture (cs.AR). ArXiv.org.*, pp. 1-33, 2007. Project license No. TXU001347562, Library of Congress, USA, ©2007.
[24] or see http://en.wikipedia.org/wiki/Solar_flare
[25] Abhyankar, K.D. (1977). "A Survey of the Solar Atmospheric Models". *Bull. Astr. Soc. India* **5**: 40–44. http://prints.iiap.res.in/handle/2248/510.
[26] S. K. Solanki et al. (1994). "New Light on the Heart of Darkness of the Solar Chromosphere". *Science* **263** (5143): 64–66. doi:10.1126/science.263.5143.64. PMID 17748350. http://www.sciencemag.org/cgi/content/abstract/263/5143/64.
[27] Gravity Assist, see for instance: http://en.wikipedia.org/wiki/Gravity_assist Accessed Sept. 2010.
[28] *Redirected Articles on*: Amdahl's law and supercomputers, available at: http://en.wikipedia.org/wiki/Supercomputer#cite_note-3
[29] D. P. Rodgers (June 1985). "Improvements in multiprocessor system design". *ACM SIGARCH Computer Architecture News archive* (New Yorok, NY, USA: ACM) **13** (3): 225–231. doi:10.1145/327070.327215. ISSN 0163-5964
[30] Serway, Raymond A. (1990). *Physics for Scientists & Engineers* (3rd ed.). Saunders. pp. 1150. ISBN 0030302587. http://books.google.com/?id=RUMBw3hR7aoC&q=inauthor:serway+photoelectric&dq=inauthor:serway+photoelectric.
[31] Francis W. Sears, W. Mark, Zemansky and D. Y. Hugh (1983), *University Physics*, Sixth Edition, Addison-Wesley, pp. 843-4. ISBN 0-201-07195-9.
[32] Joachim and Muehlner, "Trends in Missile and Space Radio Telemetry" declassified Lockheed report
[33] L. A. Zadeh, et al. 1996 *Fuzzy Sets, Fuzzy Logic, Fuzzy Systems*, World Scientific Press, ISBN 9810224214
[34] *Apply truth values*, Fuzzy Logic at http://en.wikipedia.org/wiki/Fuzzy_logic Accessed Sep 2010.
[35] Krasinsky, G. A.; Pitjeva, E. V.; Vasilyev, M. V.; Yagudina, and E. I. (July 2002). "Hidden Mass in the Asteroid Belt". *Icarus* **158** (1): 98–105. doi:10.1006/icar.2002.6837. http://adsabs.harvard.edu/abs/2002Icar..158...98K.
[36] E. V. Pitjeva. "High-Precision Ephemerides of Planets—EPM and Determination of Some Astronomical Constants". *Solar System Research* **39** (3): 176, 2005. doi:10.1007/s11208-005-0033-2. http://iau-comm4.jpl.nasa.gov/EPM2004.pdf.
[37] The definition of Overhead information at: http://en.wikipedia.org/wiki/Overhead_information, Accessed Sep. 2010.
[38] ISO/IEC 9126-1:2001 *Software engineering — Product quality — Part 1: Quality model*, ISO 9126: The Standard of Reference 2001 or see *Information technology - Software Product Evaluation - Quality characteristics and guidelines for their use* – 1991, at http://www.cse.dcu.ie/essiscope/sm2/9126ref.html, Accessed Sept. 2010.
[39] Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences. May 14, 2002.
[40] Holland, J.H.; Miller, J.H.; (1991). "Artificial Adaptive Agents in Economic Theory". *American Economic Review* **81(2)**: pp. 365–71.
[41] Application of Agent Technology to Traffic Simulation. United States Department of Transportation, May 15, 2007.
[42] "Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual". http://developer.intel.com/design/pentiumii/manuals/243191.htm. Retrieved 2007-07-13.
[43] http://www.assembly.happycodings.com/index.html
[44] Multi Core Technology, Intel® Corp, technology, at http://www.intel.com/multi-core/ or see http://en.wikipedia.org/wiki/Multi-core_processor Accessed Sep. 2010.
[45] A. H. Watson, T. J. McCabe, and D. R. Wallace (Ed.), "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," *Simplified complexity calculation*. National Inst. of Stand. and Tech. Pub., USA , pp. 23–29, 1996.
[46] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*: Vol. SE-2, No. 4, pp. 308–320, 1976.
[47] Hot carriers or hot electrons at: http://www.siliconfareast.com/hotcarriers.htm Accessed Oct. 02, 2010.
[48] J. P. Hayes, T. Mudge, Q. F. Stout, S. Colley, and J. Palmer, A microprocessor-based hypercube supercomputer. *IEEE Micro* 6, 5 (Oct. 1986), 6-17. DOI 10.1109/MM.1986.304707
[49] M. Svahnburg, Lecture Notes on Software Architecture and Quality. Blekinge Institute of Technology,

Karlskrona, Swede, Sept. 2010.
[50] M. J. Murdocca and V. P. Heuring, *Communication Errors and Error Correcting Codes*, Principles of Computer Architecture, Addison-Wesley Longman Publishing Co., Inc, pp. 358 - 369, 2000.
[51] *Hypercube Connectivity within ccNUMA Architecture*, Silicon Graphics Origin 2000, or see, http://www.risc.jku.at/education/courses/ws2000/intropar/origin/hypercube.pdf Accessed Oct. 2010.
[52] T. Theil, "The Design of the Connection Machine," *Design Issues*, Volume 10, Number 1, Spring 1994.
[53] Blob (computing) at http://en.wikipedia.org/wiki/Blob_(computing) Accessed Oct. 2010
[54] E. W. Weisstein, "Asymptote." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/Asymptote.html, Accessed Oct. 2010.
[55] J. L. Gustafson, "Increasing Hypercube Communications on Low-Dimensional Problems", Floating Point Systems, Inc. 1986, or see http://johngustafson.net/pubs/pub12/Knoxville.pdf Accessed, Oct 2010.
[56] B. Elbert, "Satellite Data Communications using VSAT Systems – Extending IT Networks to the Global Context", *Application Technology Strategy, Inc.* 2001-2010 at: http://www.applicationstrategy.com/VSAT_networks.htm Accessed Oct 2010.