ソフトウェア工学　90−10
（１９９３．２．４）

組み込み型実時間制御ソフトウェア・ファミリのための概念モデル

ベイッコ・セッパネン　　松本 吉弘

京都大学 工学部 情報工学教室
京都市左京区吉田本町

今日、多くの産業製品はその制御を実時間処理むけの組み込み型コンピュータに頼っている。７０年代にマイクロプロセッサ制御による製品が発表されて以来、これら組み込みコンピュータ・システム向けのソフトウェアには多大な進歩が見られ、累積資産も多い。そのため、これらソフトウエアの保守と再構築とを通して顧客ニーズやハードウエア技術の変化に追随していくことが重要である。そこで本稿では、組み込み型コンピュータシステムの 製品ファミリ全体を対象としたソフトウエア開発方式を提案する。この方式では、ファミリに属す個々の製品間の類似性や差別化を考慮したソフトウエアの概念モデル化を行なう。さらに、本稿では携帯電話ファミリへの適用例を通して 提案方式の有用性を述べる。

CONCEPTUAL MODELING OF FAMILIES OF REAL-TIME EMBEDDED SOFTWARE SYSTEMS

Veikko SEPPÄNEN[1]　　Yoshihiro MATSUMOTO

Kyoto University, Department of Information Science
Sakyo, Kyoto 606-01, Japan

Real-time embedded computer systems have become essential controlling parts of modern industrial products. Since the seventies, when the first microprocessor-controlled producs were introduced, considerable developments have taken place in the production of software for embedded computer systems. Several generations of such systems already exist and need to be maintained and re-engineered by the manufacturers, in order to respond to the global competition, the rapid changes of the available computer technology and the changing customer needs. The methods and tools used for the production of embedded software must be able to capture and utilize the similarities among and the differences between the concepts of *families of embedded systems*, rather than individual systems only. This paper discusses an ongoing real-life case study on the modeling of such system families.

[1]On leave from the Computer Technology Laboratory of the Technical Research Centre of Finland (VTT), P.O. Box 201, SF-90571 Oulu, FINLAND.

1. Conceptual models of computer system families

This short paper discusses conceptual analysis and modeling of families of real-time, embedded computer systems. In particular, we address both the reusability and traceability of concepts in different modeling viewpoints, such as static system structures, dynamic behaviour, data transformation functions and performance characteristics that have been found useful for describing real-time systems.

Most existing real-time system development methods do not fully integrate different modeling viewpoints together, to the degree that all concepts in different viewpoints and levels of detail or abstraction would be traceable to other relevant concepts. This may, in practice, prevent a person interested in some particular aspects of a system from focusing on a consistent and complete set of information about those aspects. End-users and software designers are typical examples of persons interested in quite specific aspects of computer-based systems. The respective aspects, when modeled, should share the same semantics as parts of the same system, although they must usually be presented using very different syntactical forms. Moreover, most current methodologies fail in supporting reusability in families of similar types of systems, although many real-time embedded computer systems version due to varying standards, implementation technologies and usage environments of the products in which they are incorporated. Cellular mobile telephone control systems that we use as a real-life case study example illustrate very well such versioning.

Therefore, it is at present difficult to reuse even minor parts or trace the consequences of even the simplest modifications of a real-time computer system. The end-user reference manual of digital GSM mobile telephones we have analyzed and the corresponding technical documentation of the real-time control system of the phones support our claim. There are only a few traceable links between system specifications, designs and implementations – not to speak of the users reference manual. We believe that one of the main reasons for this state of the matters is that real-time computer system engineers at present only implicitly maintain a common *conceptual model* of the kinds of systems they are developing. We propose that such a model is made explicit for a family of similar systems and reused as the source of *projections* that describe particular systems from different perspectives, for the needs of the people by whom the systems are developed and used.

2. Viewpoints in conceptual modeling

We have done conceptual modeling for a family of real-time, embedded computer systems by following the Kyoto Development Method [MAT92] and viewing an embedded system in the larger context illustrated in Figure 1. One of the essential concerns in this context, as opposed to most existing real-time system development methodologies, is the environment in which the computer system is incorporated. The environment usually includes several dimensions, such as the management of the usage of the product controlled by the embedded system (for example, a cellular network operator), the usage of the product (for example, a person making a mobile phone call), the non-computerized electromechanical parts of the product (such as a keyboard) and the computer system interface (such as a serial interface controller), in addition to the embedded computer system itself that consists of both hardware and software. The concepts of this complex whole should be mutually consistent and traceable, on the basis of a common conceptual model. For example end-users may, however, see only a limited set of aspects of the whole, included in "user's projections".

Within the limits of this paper, we now summarize the fundamental concerns involved in the construction of conceptual models, and then discuss the projections of such models. Our approach to conceptual modeling is based on the integration of static, dynamic, functional and performance-related viewpoints and model execution scenarios. They determine together the semantics of the conceptual model of a family of real-time embedded systems, incorporated in electromechanical products being used and managed by people or other systems.

2.1 Static modeling

The static object-based modeling viewpoint is used for organizing concepts in the dynamic, functional and performance-related viewpoints of real-time systems. One of the main reasons is to obtain a stable enough and reusable framework for describing system families, using specialization–generalization and aggregation for managing the whole of a real-time embedded computer system and the environment in which it is incorporated.

Several types of object-based and object-oriented formalisms are available for modeling static system concepts, and some of them are supported by the existing real-time CASE tools. We have applied, combined and extended some of the object-based modeling notations proposed in [RUM91, SHL92]. We use both object-attribute-relationship models and specific kinds of object communication models for describing static system structures. The former are essential for showing *is a* and *part of* types of specialization and aggregation associations in a model of a family of similar types of real-time systems. They thus lay out a foundation for the reusability of a conceptual model. We explicitly associate dynamic, functional and performance characteristics to static system entities. This makes it is possible to identify reusable patterns of the dynamic behaviour of objects at different levels of abstraction, as well as to reuse functions needed by several objects. Object communication models are needed, in practice, for modeling various ways of realizing the relationships between system entities. They make explicit the often complex exchange of data and control by messages, shared data, call parameters, interrupts, etc. mechanisms.

In the case of our example system family, we organize its atomic objects into the more abstract concepts of layers, subsystems and system units. Figure 2 shows as an example a set of static concepts of a family of digital GSM mobile telephone control systems ("OHS5. HC Handset Control Unit"), including one layered subsystem ("OHC1. CS Cellular Services Subsystem"). In the case of mobile telephone control systems, we found out that the architecture of units and subsystems is quite well-established, but very seldom explicitly documented and reused as the architecture of a problem domain [SEP92]. Moreover, although layering is nowadays often used for specifying and implementing telecommunication protocols, such routinely used real-time system development methodologies as RTSA [WAR85] completely ignore it. Obviously, they cannot be very well-tuned for modeling layered system architectures. We describe the objects of a real-time embedded system as parts or aggregates of parts of certain types of products (as "OGP4. HS Handset" in Figure 2). System objects may be interfaced with parts of the same aggregate (for example, the "OHC2. UF User Interface Subsystem" in Figure 2 is connected via "OHI5. KI Keyboard Interface" with "OHKB0. KE Key") or with other parts of the product (for example, "OHC1. CS Cellular Services Subsystem" is associated with "OGP2. SC Smart Card" via "OHI4. SN Smart Card Interface" in Figure 2).

2.2 Dynamic, functional and performance modeling

We use statecharts-style structured state transition diagrams for describing the dynamics of objects along the lines of [RUM91] and action dataflow diagrams according to [SHL92] for describing their data transformation functions. In the case of real-time, embedded computer systems, the performance constraints that need to be taken into account are most often related to timing requirements and restrictions of the physical implementation resources. We usually associate such constraints with the concepts of the dynamic and functional modeling viewpoints. For example, the key "OCK2. PW Power Switch" is an instance of the "OHKB0. KE Key" object shown in Figure 2. There is a requirement for pressing that key for at least 300 milliseconds to switch the power off, if the power has been turned on. This requirement is associated with the object "OUF2. PR Power Switch Controller" (not shown in Figure 2) that belongs to the object "OHC2. UF User Interface Subsystem". If it is met at some point, the object "ODM1. PS Power Supply" of "OHC3. DM Device Monitor Subsystem" is informed, so that it can gracefully switch off "OGP2. SC Smart Card", "OHS4. HI Handset Control Unit Interface", "OHS5. HC HAndset Control Unit", and finally the physical power source 'OGP4. PO Power Source".

We integrate and trace together concepts within the modeling viewpoints at different levels of abstraction. In its simplest form this can be done graphically, by the appropriate naming of specialized or aggregated concepts and by adding elementary trace information. In Figure 2, for example, the name of the relationship "RUL7. Commands" indicates the target entity, the layer "OCS1. UL Upper Layer". This layer is modeled as a part of the subsystem "OHC1. CS Cellular Services Subsystem", which in turn belongs to system unit "OHS5. HC Handset Control Unit". For example, "OHC1." in the name of the subsystem indicates that it is a numbered part of another object whose code is "HC". In the object communication model (not shown in Figure 2), the name "SUL4. [RUL7.] Command received" of a message-based communication item would tie one of the stimuli of the subsystem to the relationship RUL7. In the same way, the name "SSE4. [RSE1.] [SUL4.] Search for neworks" would be used for a stimulus from the user interface subsystem to one of the objects belonging to "OCS1. UL Upper Layer". This helps tracing the stimulus to the layer-level stimulus SUL4., but also to the corresponding relationship RSE1. at the object level.

As a more advanced form of integration and traceability, we construct execution scenarios in which the execution principles of the modeled embedded computer systems are made explicit and not hidden inside a CASE tool. We have been experimenting both with graphical and textual execution scenarios, starting from the kinds of descriptions discussed for example in [SHL92] (thread of control models) and in [RUM91] (scenarios and event traces).

3. Projections of a conceptual model

A projection is a limited perspective to a conceptual model. Its purpose is to describe a specific system (as opposed to a family of systems) for the needs of particular persons dealing in certain roles with that system. A projection may be described using specific syntactic notations, but so that the semantics of the conceptual model are preserved. Therefore, different projections are made traceable to each other only via the common conceptual model. This principle fundamentally differs from the traditional approach to trying to ensure traceability from embedded system specifications to their implementations. As indicated by our case study, such an approach is often doomed to fail due to several conceptual gaps within the system development process, in addition to gaps between user documents and design descriptions.

With regard to the dynamic modeling viewpoint, one of the key principles needed for deriving user's projections is to make a clear difference between the stimuli and responses processed by the embedded computer system and the corresponding usage events and effects in the external environment of the computer system. As illustrated in Figure 1, this means that the dynamics of the conceptual model are projected to the *usage environment* and to parts of the *unintelligent electromechanical product environment* of the product being controlled by the computer system.

The former, as we have modeled it, includes the model of the dynamic state behaviour of the user himself or herself, in addition to the description of usage events and effects. The latter includes dynamic models of the input and output devices accessible by the user, as well as of the software-based entities managed by the user. Moreover, information provided by the user or for the user via the input/output devices must be depicted. Since it would be cumbersome to deal with the physical user interface entities (such as "a key that has the label C") we define the logical counterparts of each device (for example, "Cancel Key") and use the latter when modeling the dynamic behaviour of each entity. A simple way for associating the two together is visual correspondence between the physical and logical entities of the user interface. A *usage scenario* corresponds to the execution scenario of the conceptual model, from which the user's projection has been derived. In usage scenarios we, however, show only the intended use of the computer-controlled product, the exceptional situations being included.

4. Summary

In this paper we claim that the state-of-the-practice in modeling real-time embedded software systems lacks effective means for ensuring traceability within the whole of a computer system and its environment, as well as for promoting the reuse of system structures, patterns of dynamic behaviour and data processing functions similar to several members of the same system family. The paper briefly demonstrates how the construction of conceptual models of families of real-time systems and the generation of end-user projections from such models support both traceability and reusability. A more comprehensive discussion of the research, with an extensive set of examples taken from the conceptual model of a family of digital mobile telephone control systems is provided in [SEP93].

5. References

[MAT92] MATSUMOTO, Y. *KDM: Kyoto Software Development Method.* Working paper, Kyoto University, Department of Information Science, 1992. 27 p.

[RUM91] RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F. AND LORENSEN, W. *Object-Oriented Modeling and Design.* Englewood Cliffs, New Jersey, Prentice-Hall, 1991. 500 p.

[SHL92] SHLAER, S., MELLOR, S.J. *Object Lifecycles: Modeling the World in States.* Englewood Cliffs, New Jersey, Prentice-Hall, 1992. 243 p.

[SEP92] SEPPÄNEN, V. *Acquisition, organisation and reuse of software design knowledge.* Software Engineering Journal, vol. 7, no. 4, July 1992, pp. 238 - 246.

[SEP93] SEPPÄNEN, V. *Flexible software manufacturing: Modeling of factory products.* Kyoto University, Department of Information Science,Technical Report KUIS-93-0001 (ISSN 0918-4163), February 1993. 49 p. + app. 53 p. (to appear)

[WAR85] WARD P.T., MELLOR S.J. *Structured Development for Real-Time Systems.* New York, Yourdon Press, 1985, Vol. I-III.

Figure 1. The modeling context of embedded systems

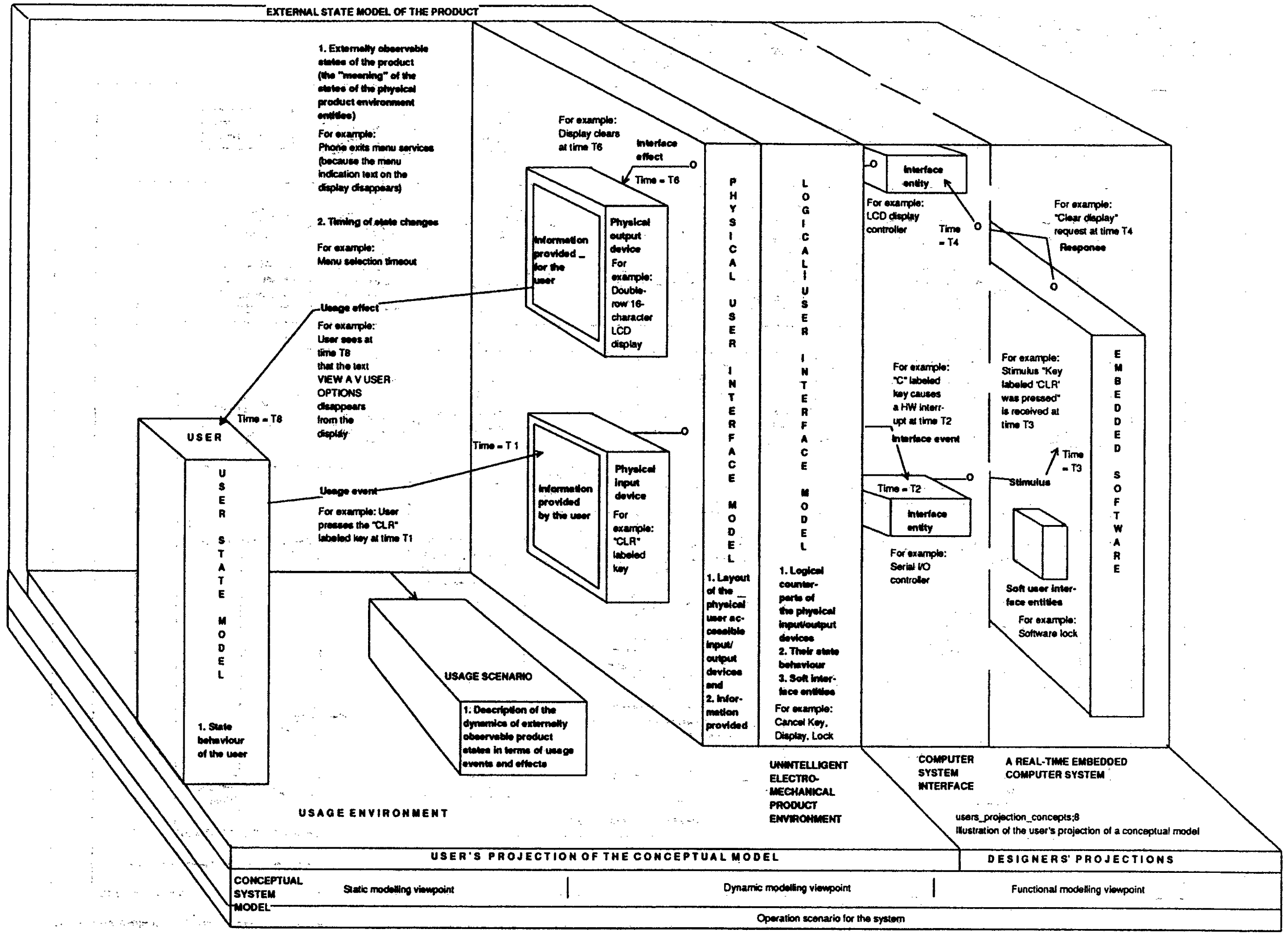
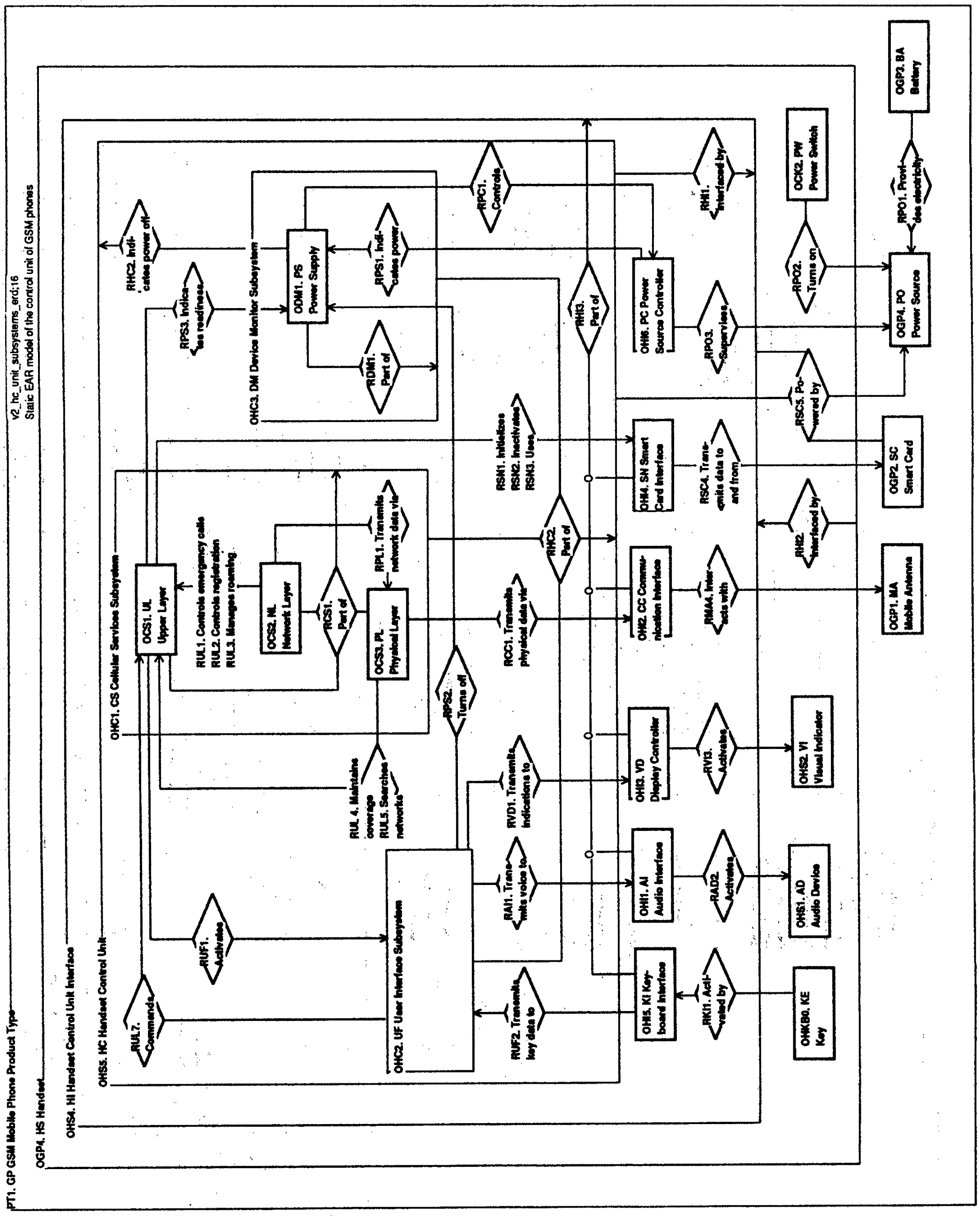

EXTERNAL STATE MODEL OF THE PRODUCT
1. Externally observable states of the product (the "meaning" of the states of the physical product environment entities)
For example: Phone exits menu services (because the menu indication text on the display disappears)
2. Timing of state changes
For example: Menu selection timeout
Usage effect
For example: User sees at time T8 that the text VIEW A V USER OPTIONS disappears from the display
Time = T8
USER
USER STATE MODEL
1. State behaviour of the user
Usage event
For example: User presses the "CLR" labeled key at time T1
For example: Display clears at time T6
Interface effect
Time = T6
Information provided for the user
Physical output device
For example: Double-row 16-character LCD display
Time = T 1
Information provided by the user
Physical input device
For example: "CLR" labeled key
USAGE SCENARIO
1. Description of the dynamics of externally observable product states in terms of usage events and effects
PHYSICAL USER INTERFACE MODEL
1. Layout of the physical user accessible input/output devices and
2. Information provided
LOGICAL USER INTERFACE MODEL
1. Logical counterparts of the physical input/output devices
2. Their state behaviour
3. Soft interface entities
For example: Cancel Key, Display, Lock
Interface entity
For example: LCD display controller
Time = T4
For example: "C" labeled key causes a HW interrupt at time T2
Interface event
Time = T2
Interface entity
For example: Serial I/O controller
For example: "Clear display" request at time T4
Response
For example: Stimulus "Key labeled 'CLR' was pressed" is received at time T3
Time = T3
Stimulus
Soft user interface entities
For example: Software lock
EMBEDDED SOFTWARE
USAGE ENVIRONMENT
UNINTELLIGENT ELECTRO-MECHANICAL PRODUCT ENVIRONMENT
COMPUTER SYSTEM INTERFACE
A REAL-TIME EMBEDDED COMPUTER SYSTEM
users_projection_concepts;8
Illustration of the user's projection of a conceptual model
USER'S PROJECTION OF THE CONCEPTUAL MODEL
DESIGNERS' PROJECTIONS
CONCEPTUAL SYSTEM MODEL
Static modelling viewpoint
Dynamic modelling viewpoint
Functional modelling viewpoint
Operation scenario for the system

Figure 2. A static model of the control system of a GSM phone



v2_hc_unit_subsystems_erd;16
Static EAR model of the control unit of GSM phones
PT1. GP GSM Mobile Phone Product Type
OGP4. HS Handset
OHS4. HI Handset Control Unit Interface
OHS5. HC Handset Control Unit
OHC1. CS Cellular Services Subsystem
OHC2. UF User Interface Subsystem
OHC3. DM Device Monitor Subsystem
ODM1. PS Power Supply
RDM1. Part of
RPS1. Indicates power
RPS2. Turns off
RPS3. Indicates readiness
RHC2. Indicates power off
RPC1. Controls
RHI1. Interfaced by
RHI2. Interfaced by
RHI3. Part of
RHC2. Part of
OCS1. UL Upper Layer
OCS2. NL Network Layer
OCS3. PL Physical Layer
RCS1. Part of
RUL1. Controls emergency calls
RUL2. Controls registration
RUL3. Manages roaming
RUL 4. Maintains coverage
RUL5. Searches networks
RUL7. Commands
RUF1. Activates
RUF2. Transmits key data to
RPL1. Transmits network data via
RCC1. Transmits physical data via
RSN1. Initializes
RSN2. Inactivates
RSN3. Uses
RVD1. Transmits indications to
RAI1. Transmits voice to
OHI6. PC Power Source Controller
OHI4. SN Smart Card Interface
OHI2. CC Communication Interface
OHI3. VD Display Controller
OHI1. AI Audio Interface
OHI5. KI Keyboard Interface
RPO3. Supervises
RSC4. Transmits data to and from
RMA4. Interacts with
RVI3. Activates
RAD2. Activates
RKI1. Activated by
OCH2. PW Power Switch
RPO2. Turns on
OGP3. BA Battery
RPO1. Provides electricity
OGP4. PO Power Source
RSC5. Powered by
OGP2. SC Smart Card
OGP1. MA Mobile Antenna
OHS2. VI Visual Indicator
OHS1. AD Audio Device
OHKB0. KE Key