QSEE
Qualidade do Software Embarcado em Aplicações Espaciais

Documento de Projeto do Software

Q00-DPSw-v05 15 de Dezembro de 2006

QSEE
Qualidade do Software Embarcado em Aplicações Espaciais

Data: 15/12/2006	**Nome do Arquivo:** QSEE_Documento de Projeto de Software_v05.pdf

Documento de Projeto do Software

Preparado Por:	Wendell Pereira da Silva	**Data e Visto**
Aceito Por:	Valdivino Alexandre de Santiago Júnior	**Data e Visto**
Aprovado Por:	Maria de Fátima Mattiello-Francisco	**Data e Visto**

Para												
Aplicação aos												
Modelos												

Documento Configurável	**Por**: Valdivino Alexandre de Santiago Júnior	**Desde:**22/02/2006

ÍNDICE

LISTA DE ACRÔNIMOS

AR	*Acceptance Review*
bps	bits por segundo
CASE	*Computer Aided System Engineering*
CDR	*Critical Design Review*
CEU	*Central Electronics Unit*
CVS	*Concurrent Versioning System*
DPS	Documento de Projeto do Software
EPP	*Event Pre-Processor*
FSM	*Finite State Machine*
GPIO	*General Purpose I/O*
GSE	*Ground Support Equipment*
HXI	*Hard X-Ray Camera*
Hz	Hertz
IRD	*Interface Requirements Document*
ICD	*Interface Control Document*
MCU	*Microcontroller Unit*
NA	*Not Applicable*
OBDH	*On-Board Data Handling*
PDC	*Payload Data Handling Computer*
PDR	*Preliminary Design Review*
QR	*Qualification Review*
QSEE	Qualidade do Software Embarcado Para Aplicações Espaciais
RB	*Requirements Baseline*
SEU	*Single Event Upset*
SFR	*Special Function Register*
SRR	*System Requirements Review*
SWPDC	Software Piloto Embarcado no PDC
SXI	*Soft X-Ray Camera*
TBC	*To Be Confirmed*
TBD	*To Be Determined*
TBR	*To Be Resolved*
TBS	*To Be Supplied*
UTC	*Universal Coordinated Time*
UML	*Unified Modeling Language*
V	Volt
WBS	*Work Breakdown Structure*

LISTA DE FIGURAS

1 INTRODUÇÃO

Este documento descreve o projeto arquitetural detalhado do software e os artefatos utilizados no projeto detalhado do SWPDC, desenvolvido no contexto do projeto QSEE – Qualidade do Software Embarcado em Aplicações Espaciais. O projeto das interfaces internas ao sistema, por exemplo: *drivers* e *buffers* de dados estão incluídos neste documento. O DPSw é elaborado pelo fornecedor e disponibilizado para acompanhamento e revisão do cliente conforme os marcos estabelecidos no contrato. Este documento está em consonância com os requisitos e fases especificadas nos documentos Requisitos de Base (AD1), Plano de Validação e Verificação Independente (AD2) fornecidos pelo cliente, e segue o Plano de Desenvolvimento do Software (AD4) elaborado pelo fornecedor.

O projeto do software tem o objetivo de descrever como a solução data pela especificação técnica (AD5), ou seja, os requisitos funcionais e não funcionais, serão construídos em termos de artefatos de software.

2 DOCUMENTOS APLICÁVEIS E DE REFERÊNCIA

Os seguintes documentos são aplicáveis:

AD1 – Requisitos de Base - Q00-RB-v08. Projeto QSEE, Instituto Nacional de Pesquisas Espaciais, Agosto de 2006.

AD2 – Plano de Validação e Verificação Independente de Software – Q00-PVVIS-v04. Projeto QSEE, Instituto Nacional de Pesquisas Espaciais, Agosto de 2005.

AD3 – Projeto MIRAX – Monitor e Imageador de Raios X. Ciências Espaciais e Atmosféricas, Instituto Nacional de Pesquisas Espaciais. Disponível em: <http://www.cea.inpe.br/cea/satelites/mirax/miraxproject.htm>. Acesso em: Julho de 2005.

AD4 – Plano de Desenvolvimento do Software – Q00-PDSw-v06. Projeto QSEE, Instituto Nacional de Pesquisas Espaciais, Outubro de 2005.

AD5 – Especificação Técnica do Software – Q00-ETS-v04. Projeto QSEE, Instituto Nacional de Pesquisas Espaciais, Outubro de 2006.

AD6 – Protocolo de Comunicação PDC-OBDH – Q00-PPDOB-v06. Projeto QSEE, Instituto Nacional de Pesquisas Espaciais, Agosto de 2006.

AD7 – Protocolo de Comunicação PDC-EPPs – Q00-PPDEP-v05. Projeto QSEE, Instituto Nacional de Pesquisas Espaciais, Agosto de 2006.

AD8 – Manual do CVS on-line. Disponível em: <http://ximbiot.com/cvs/wiki>. Acesso em: Janeiro de 2006.

AD9 – Sistema de documentação de código-fonte C/C++, Java e IDL, *Doxygen*. Disponível em: <http://www.doxygen.org/>. Acessado em Janeiro de 2006.

Os seguintes documentos são referências:

RD1 – Software Engineering: A Practitioner´s Approach. Roger Pressman, McGraw-Hill, 2001.

RD2 – Adapting Function Points to Real-Time Software. St-Pierre et al., IFPUG 1997 Fall Conference. Disponível em < http://www.lrgl.uqam.ca/publications/presenta/lrgl-1997-020b.pdf>. Acesso em: Setembro de 2005.

RD3 – Recommended C Style and Coding Standards. Bell Labs, University of Toronto, EECS/UC Berkeley, CS&E/University of Washington, SoftQuad Incorporated/Toronto. Disponível em: < http://www.psgd.org/paul/docs/cstyle/cstyle.htm>. Acesso em: Setembro de 2005.

RD4 – ECSS Space Project Management: Configuration Management. ECSS-M-40A, Abril de 1996.

RD5 – ECSS Space Product Assurance: Software Product Assurance. ECSS-Q-80B, Outubro de 2003.

RD6 – The C Kernel Pages: C Style and Coding Standards. Disponível em: http://home.hetnet.nl/~p_vd_vlugt/. Acesso em: Janeiro de 2006.

RD7 – UML Guia do Usuário, Booch, Humbaugh, Jacobson, Ed.Campus, 2000.

RD8 – UML 2.0, The Current Official Specification. Disponível em: <: http://www.uml.org/>. Acesso em Fevereiro de 2006.

RD9 - Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, et.al. Addison-Wesley, 1995.

RD10 - uPSD33xx Turbo Series – Fast 8032 MCU with Programmable Logic, STMicroeletronics, Janeiro-2005, (*Product Datasheet*).

3 ESTRATÉGIA DE DESENVOLVIMENTO

O objetivo deste capítulo é descrever a estratégia de desenvolvimento adotada pela equipe do fornecedor INPE para construir o código fonte do SWPDC de acordo com o modelo detalhado dos componentes descritos no Capítulo 4.

Do ponto de vista do desenvolvimento do software, os modelos descritos no documento de projeto de software evoluem na medida em que as soluções arquiteturas são codificadas. Assim, nem tudo que foi modelado inicialmente vai refletir, de imediato, o código fonte do SWPDC, e vice e versa. A estratégia de desenvolvimento adotada deixa duas atividades bem definidas:

1. **Engenharia de produção**: alterações no modelo (classes, estados, atividades) são codificadas, num dado instante, manualmente pelo **desenvolvedor**.
2. **Engenharia reversa**: alterações no código fonte não previstas nos modelos são, em algum momento, recuperadas manualmente e analisadas pelo **arquiteto**.

3.1 MAPEAMENTO DO PROJETO PARA ARTEFATOS DE DESENVOLVIMENTO

Este projeto de software não contempla o uso de ferramentas CASE capazes de realizar automaticamente as atividades de engenharia de produção e reversa. Portanto, faz-se necessária alta interatividade entre desenvolvedor e arquiteto, uma vez que tais atividades são executadas manualmente, porém com o auxílio de IDEs e ferramentas de modelagem UML. Além disso, uma vez que os modelos são alterados, seja para refletir melhor o código fonte, seja para melhorar a arquitetura ou detalhar melhor certos aspectos de projeto, o documento de projeto de software também deverá ser atualizado.

De modo geral, realizam-se, com certo grau de paralelismo, as seguintes atividades:

1. O **arquiteto** mapeia os requisitos técnicos para o modelo na forma de diagramas estruturais e comportamentais da UML, mantendo o documento de projeto de software.
2. O **desenvolvedor** produz artefatos físicos de construção (i.e. arquivos de código fonte, arquivos de projeto de IDE, arquivos de configuração, testes unitários e scripts de construção) baseado nos modelos UML.

Para nortear a construção do código a partir da leitura dos modelos, alguns procedimentos foram definidos e estão detalhados no APÊNDICE B.

3.2 PRODUÇÃO DO CÓDIGO FONTE

A princípio, pode-se achar estranha a idéia de usar modelos em UML e projeto de software orientado a objetos, para um software embarcado numa plataforma de hardware baseado em microcontrolador, escrito em linguagens de programação como C e Assembly que não suportam diretamente o paradigma orientado a objetos. Apesar disso, práticas foram adotadas para tornar o projeto de software gerenciável e razoavelmente fiel ao código-fonte produzido, mesmo manualmente:

Projeto:

1. Não abusar das hierarquias de classes / interfaces.
2. Priorizar herança de interface em vez da de comportamento (subclasses)
3. Explicitar acessos aos métodos: operações públicas ficam em interfaces, operações locais ficam privadas às classes.

4. Em função das limitações de recursos de hardware da plataforma (memória, CPU, I/O), priorizar o uso de estruturas estáticas em vez de dinâmicas; ou seja, fazendo uso de classes de instância única (*Singleton*).
5. Explicitar no modelo o mapeamento da memória.

Construção:

1. Polimorfismo com ponteiros para funções.
2. Classes são mapeadas em dois arquivos com as seguintes extensões:
   - **.h**: operações públicas e declarações de dependências externas.
   - **.c**: implementação do comportamento, especificado pela arquitetura dinâmica.
3. Interfaces geram somente arquivos **.h** e declaram dependências externas.
4. Componentes UML são mapeados em pastas (diretórios) contendo suas respectivas classes e interfaces (arquivos .asm, .c e .h).
5. Declaração de macro comandos (ou diretivas de pré-processamento) para ocultar detalhes específicos (e muitas vezes exóticos) da plataforma de hardware.
6. Coleções implementadas como vetores de tamanho fixo alocados estaticamente

Em termos de atividades dos atores principais, a Figura 3.1 apresenta uma visão geral da estratégia adotada para o desenvolvimento do SWPDC.

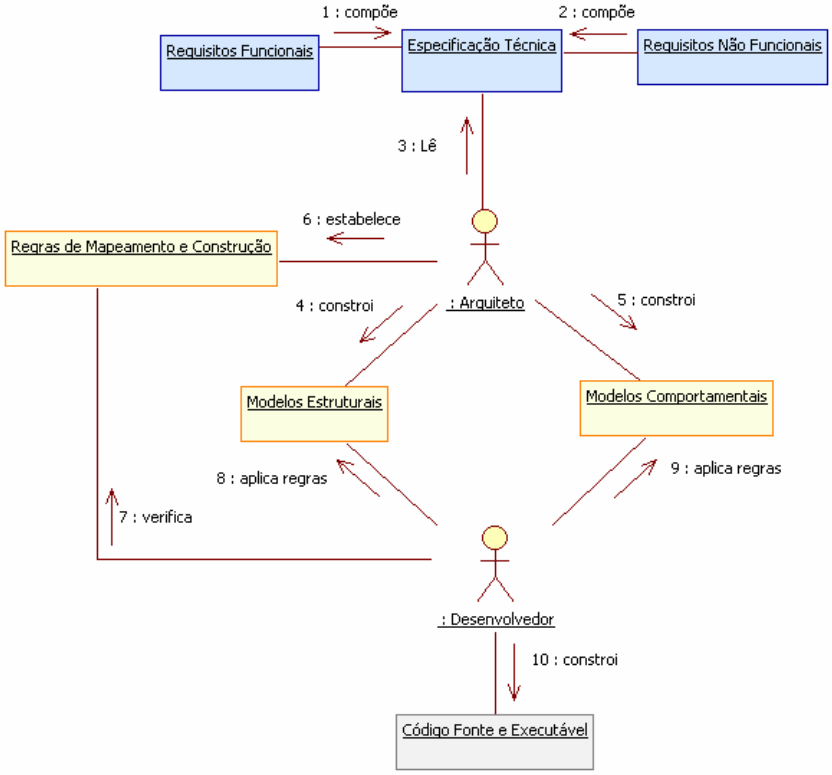

Figura 3.1: Visão geral da estratégia adotada para construção do código fonte

4 ARQUITETURA DO SOFTWARE

O software SWPDC será desenvolvido no Laboratório de Altas Energias da Coordenadoria de Ciências Espaciais e Atmosféricas (CEA), com a participação de competências na área de Verificação, Validação e Testes disponíveis na Engenharia e Tecnologias Espaciais (ETE), no INPE, em São José dos Campos/SP.

O projeto do SWPDC segue a abordagem do ciclo de vida de software em cascata. Assim, o projeto do software propriamente dito será uma evolução da solução de projeto proposta no documento de Especificação Técnica (AD5) no sentido de consolidar os modelos descritivo, conceitual e operacional do software.

4.1 CONTEXTO GERAL

O diagrama da Figura 4.1 mostra o contexto geral do SWPDC com relação ao mundo a sua volta (entidades/subsistemas externos).

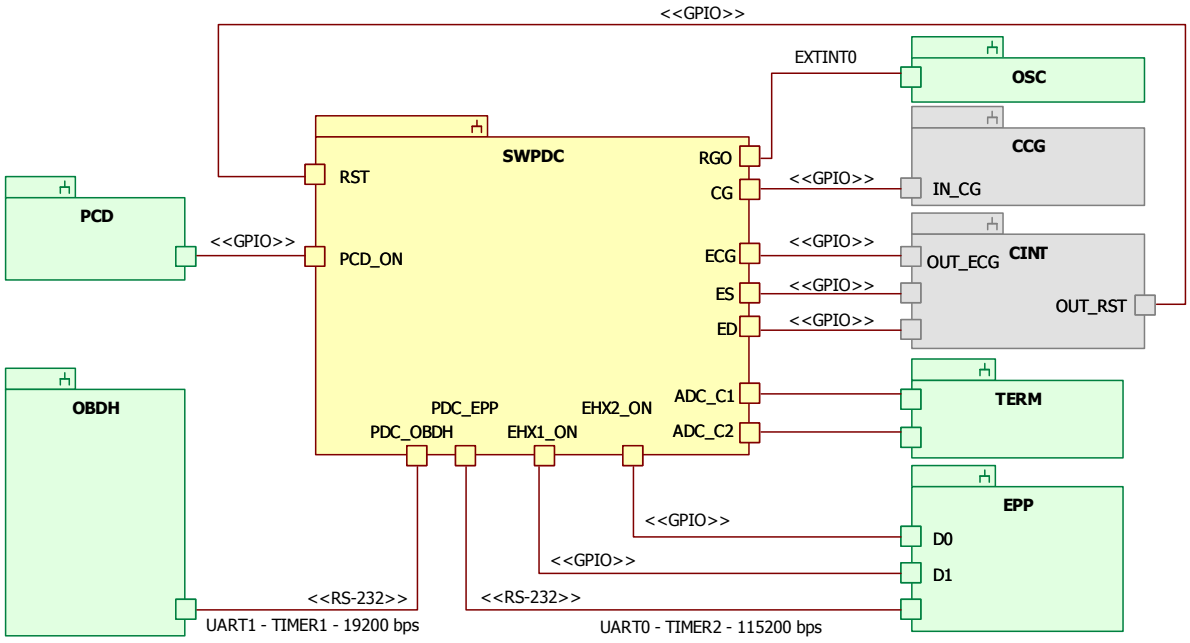

Figura 4.1: SWPDC e Subsistemas [CLS-PDC-00].

Esta visão denota o mais alto nível conceitual de modelagem adotada para o SWPDC. Este diagrama mostra as entidades externas ao SWPDC e suas interconexões (portas).

As entidades externas PCD, CCG, TERM, OSC, CINT, EPP e OBDH estão representadas como subsistemas:

- **PCD**: O sistema de condicionamento e distribuição de energia.
- **CCG**: O circuito de cão-de-guarda.
- **TERM**: Os sinais analógicos para leitura de temperaturas.

- **EPP**: O pré-processador de eventos das câmeras de raios-x duros.
- **OSC**: Oscilador externo operando a 1 KHz.
- **CINT**: Circuito externo gerador de interrupções de hardware (emulado via software para propósitos de teste).
- **OBDH**: O computador de bordo.

As portas denotam as interfaces de hardware entre o SWPDC e demais subsistemas:

- **PDC_OBDH**: Porta serial (UART/RS-232), comunicação com OBDH.
- **PDC_EPP**: Porta serial (UART/RS-232), comunicação com EPP.
- **EHX1_ON**: Comando ON/OFF EPP-HX-1.
- **EHX2_ON**: Comando ON/OFF EPP-HX-2.
- **ADC_C1, ADC_C2**: Canais analógicos para aquisição de temperaturas.
- **CG**: Atualização do cão-de-guarda.
- **ECG**: Interrupção para detecção de erro de cão-de-guarda.
- **ES**: Interrupção para detecção de erros simples.
- **ED**: Interrupção para detecção de erros duplos.
- **RST**: Interrupção para *reset* da MCU.
- **PCD_ON**: Interrupção para monitoramento do sinal de alimentação.

O mapeamento das portas será feito via registrador de função especial (SFR) da MCU, como mostra a Figura 4.2. Detalhes da MCU podem ser encontrados em (RD10). O critério de escolha de pinos da MCU foi a sua disponibilidade para ser programado de acordo com as necessidades do projeto, considerando eventuais interferências eletromagnéticas verificadas por meio de testes preliminares no modelo de engenharia (RD10).

A porta PDC_OBDH do modelo de software será realizada pelo conjunto de pinos UART1_TxD e UART1_RxD.

A porta PDC_EPP do modelo de software será realizada pelo conjunto de pinos UART0_TxD e UART0_RxD.

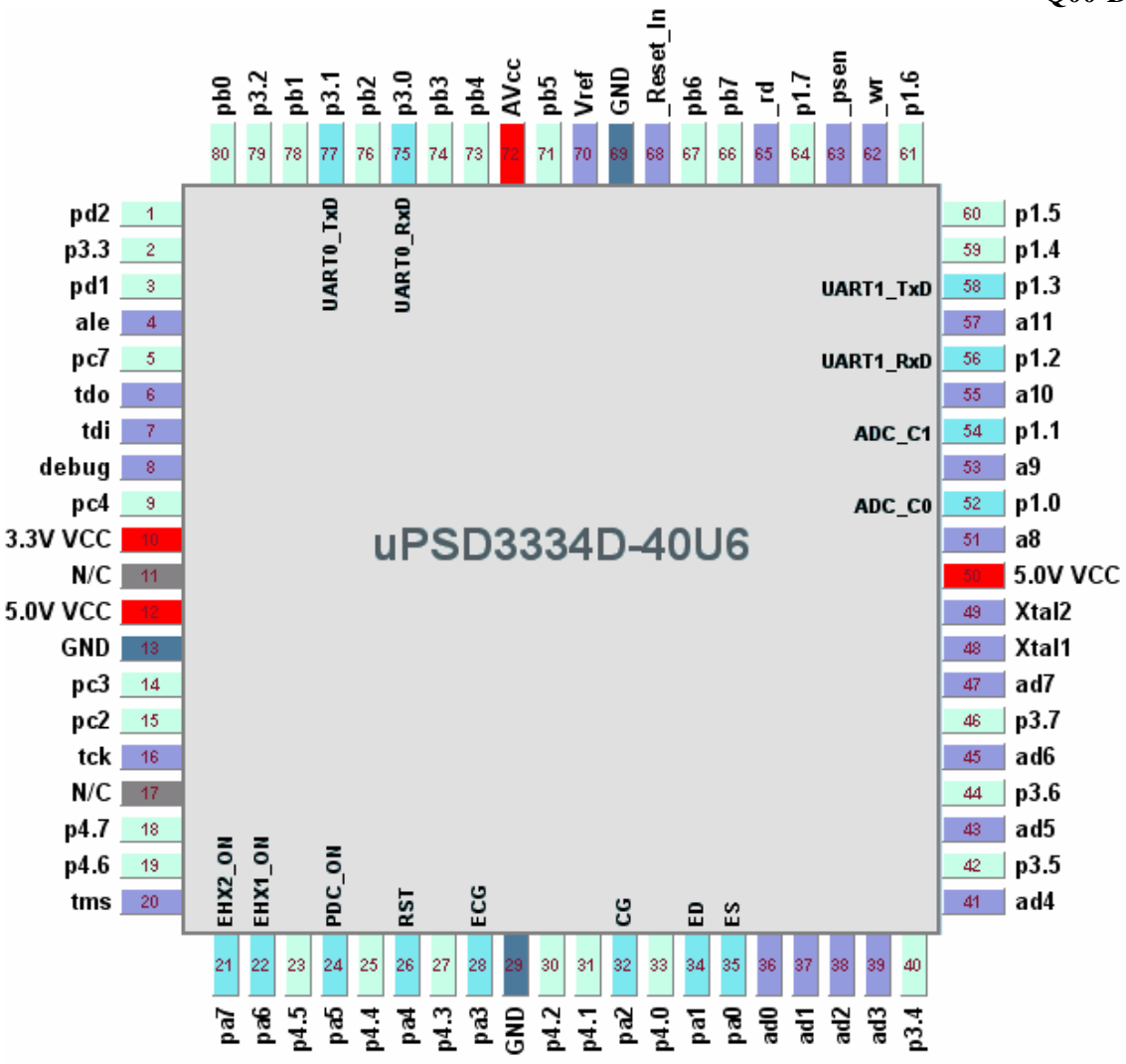

Figura 4.2: Mapeamento de pinos da MCU.

4.2 ARQUITETURA ESTÁTICA

Esta seção descreve a modelagem estrutural do SWPDC. Os modelos estão expressos na forma de diagramas da linguagem UML versão 2.0 (RD7) (RD8).

4.2.1 VISÃO EM CAMADAS

O Software Piloto (SWPDC) que será embarcado no *Payload Data Handling Computer* (PDC) é o item de software objeto deste projeto. Para cumprir as funcionalidades requeridas em (AD1) e detalhadas em (AD5), o software PDC foi decomposto em 8 categorias de serviços:

a. **Gerenciamento de Estado** – agrega os componentes que executam as mudanças de operação do software;

b. **Gerenciamento de Dados** – agrega os componentes que operam com dados científicos e de status do sistema
c. **Suporte** – compreende as funcionalidades meio para operação do software.
d. **Controle de Serviços de Aplicação** - responsável pela execução dos comandos recebidos da interface de comunicação com o OBDH chamando os componentes envolvidos na tarefa.
e. **Obtenção de dados para TM** – responsável pelo controle da aquisição dos dados de TM (dados científicos) através da interface com os EPPs.

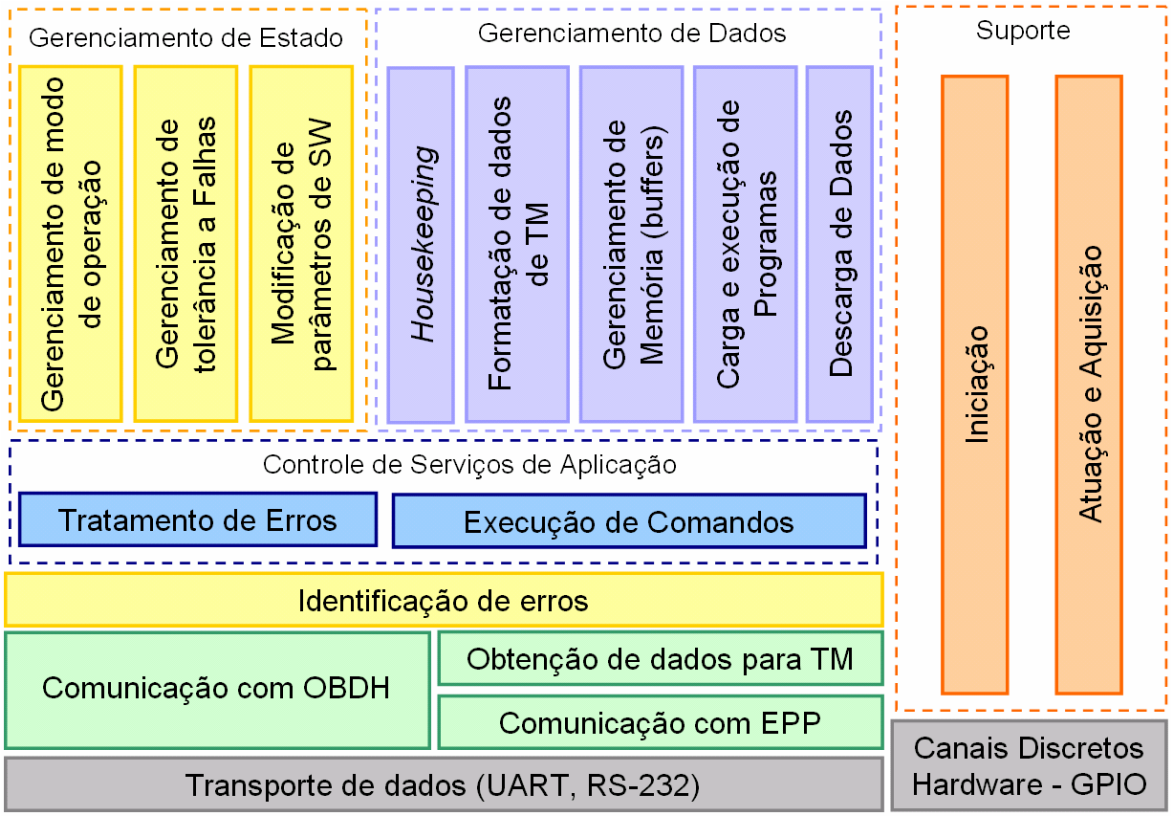

Figura 4.3: Arquitetura do Software PDC.

f. **Identificação de Erros** – responsável pela implementação dos mecanismos de apoio a tolerância à falhas.
g. **Comunicação com OBDH** – funcionalidades de acesso à interface serial RS-232 para comunicação com a entidade externa OBDH.
h. **Comunicação com os EPPs** – implementa as funcionalidades de acesso à interface serial RS-232 para comunicação com as entidades externas EPPs (H1 e H2).

As três primeiras categorias descritas acima, **Gerenciamento de Estado**, **Gerenciamento de Dados** e **Suporte** foram subdivididas da seguintes forma:

a1. **Gerenciamento de modos de operação** – responsável pela consistência dos modos de operação e chaveamento entre eles;
a2. **Gerenciamento de Tolerância a Falhas** – responsável pela tomada de decisão quando uma falha foi identificada;
a3. **Modificação de parâmetros de Software** – responsável pela alteração das tabelas de configuração do software;

b1. **Housekeeping** – efetua a coleta dos parâmetros relevantes para a monitoração e controle da saúde do instrumento.
b2. **Formatação de dados de TM** – funções de empacotamento dos dados científicos, *housekeeping*, diagnose, teste e descarga de memória para transmissão para o OBDH.
b3. **Gerenciamento de memória** – responsável pela alocação e gestão da memória de dados (*buffers*).
b4. **Carga e Execução de programa** – responsável pelo controle e checagem da consistência dos pacotes de carga de programa assim como da execução propriamente dita da carga.
b5. **Descarga de dados** – responsável pela execução do *dump* de memória, isto é, cópia de áreas de memória solicitadas e armazenamento em *buffers* para transmissão para o OBDH.
c2. **Iniciação** – efetua o boot do PDC verificando a eventual presença de um novo programa (*firmware updated*) válido carregado previamente. Faz a substituição do *firmware* atual pelo novo.
c3. **Aquisição e Atuação** – responsável pela leitura e escrita nos canais digitais (comandos ON/ OFF) e conversores AD.

A camada mais abaixo (Transporte de Dados e Canais Discretos / Hardware) foi adicionada à figura apenas a título de ilustração, pois corresponde ao nível físico (hardware), sendo acessada via registros de função especial da MCU. As camadas mais acima na Figura 4.3 representam uma visão introdutória da arquitetura estática do software SWPDC. Os detalhes relativos à implementação dos componentes dessas camadas serão detalhados no Capítulo **Erro! Fonte de referência não encontrada.**.

4.2.2 VISÃO EM COMPONENTES

Isolando o SWPDC na visão de subsistemas apresentada na Figura 4.1, a Figura 4.4 mostra um relacionamento entre os seus componentes[1] internos, por meio de suas interfaces. Os cinco componentes que constituem o SWPDC são **Serviços**, **Dados**, **Estado**, **Comunicação**, e **Suporte**.

O componente central é **Serviços**, responsável pela tomada de decisões inerentes à determinação e execução de comandos. O componente de **Comunicação** encapsula as operações que manipulam a comunicação do SWPDC com OBDH e EPP. As operações de atuam nos dispositivos de hardware ficam a cargo do componente de **Suporte**, assim como procedimentos de iniciação e funções utilitárias. A persistência dos dados gerados ou coletados pelo SWPDC, fica a cargo do componente de **Dados**. A memória de configuração, por ser mais sensível às condições operacionais do PDC, é encapsulada pelo componente **Estado** que fornece interfaces de acesso a essa memória. Além disso, **Estado** mantém um arcabouço (*framework*) para implementação simplificada de máquinas de estados finitos.

[1] Neste contexto, o termo **componente** é sinônimo de módulo, ou seja, parte do software que pode se comunicar com os demais módulos por meio de interfaces bem definidas. Um componente pode conter outros módulos (classes) que efetivamente realizam as interfaces expostas pelos componentes.

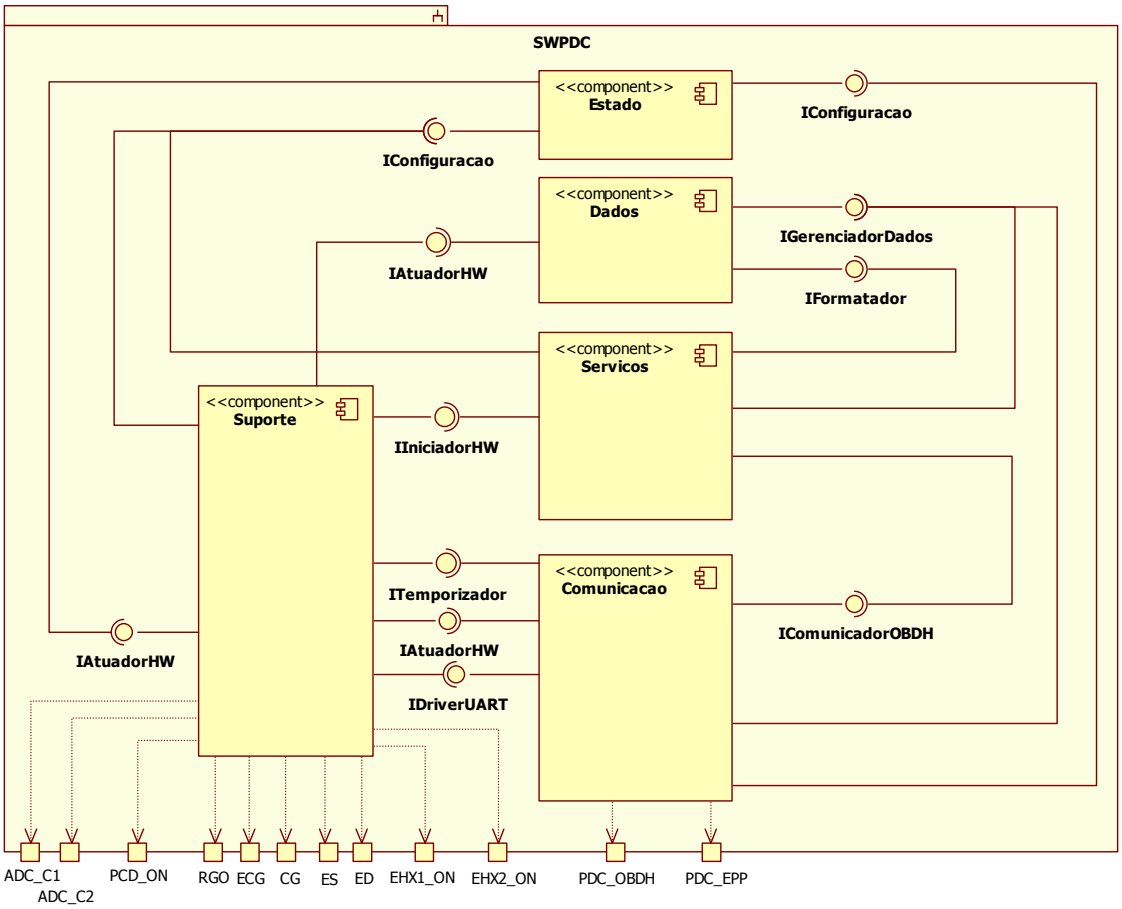

Figura 4.4: Componentes e interfaces [CLS-PDC-01].

Complementando a Figura 4.4, a visão apresentada na Figura 4.5 mostra a adoção do padrão de projeto Observador/Sujeito (RD9) para a notificação da ocorrência eventos. Assim, há a separação entre componentes geradores de eventos e aqueles que têm que ser notificados quando da ocorrência de eventos que requerem processamento em nível de serviço, como por exemplo, a identificação de erros, um novo comando pronto para processar, um agrupamento de pacote de dados (*EventPacks*) adquirido completamente, etc. Embora cada componente seja potencialmente, ao mesmo tempo, sujeito a eventos e observador de eventos, a Figura 4.5 dá uma visão geral deste esquema de notificação/tratamento de eventos.

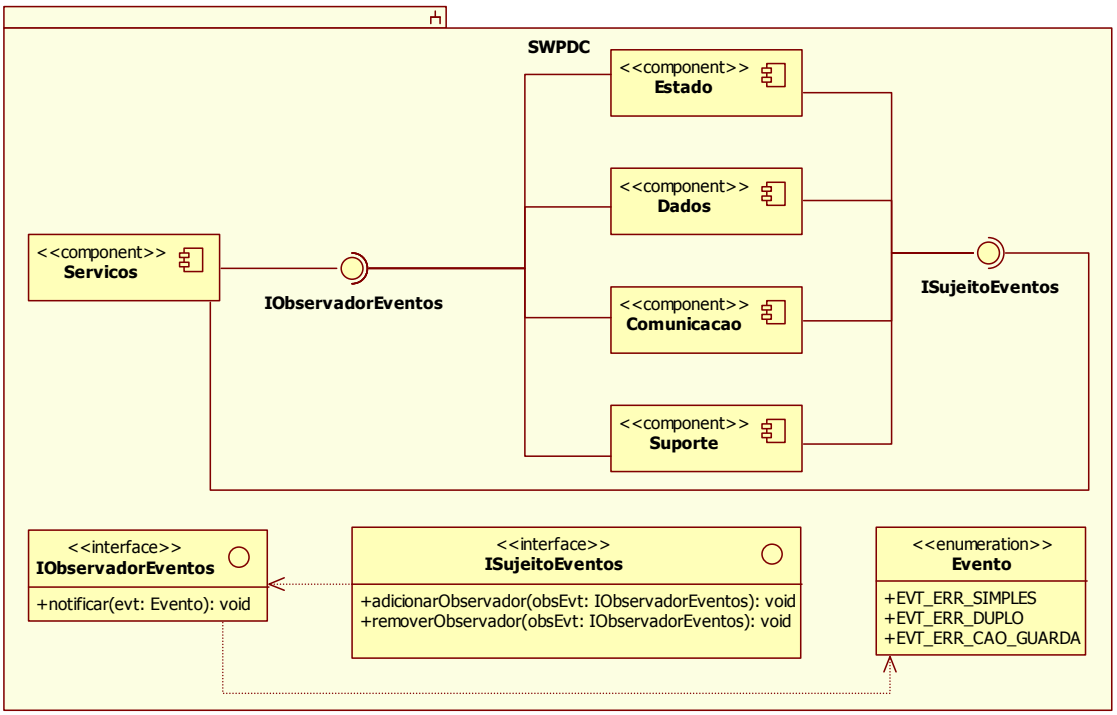

Figura 4.5: Propagação de Eventos [CLS-PDC-02].

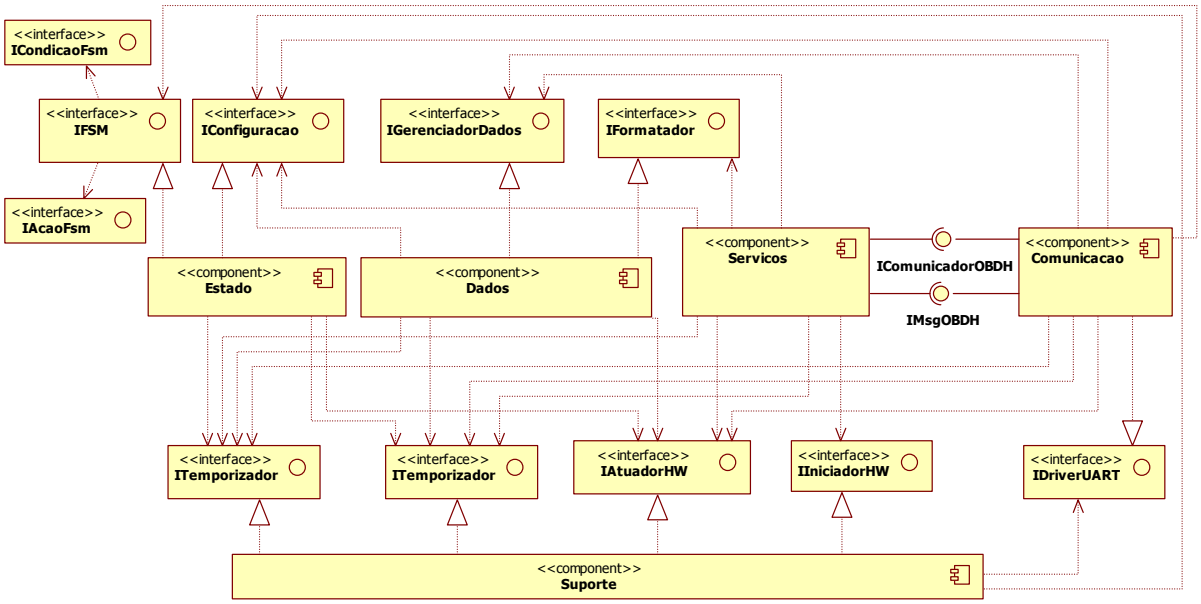

Figura 4.6: Dependência dos componentes por meio de suas interfaces [CLS-PDC-03].

A Figura 4.6 complementa a Figura 4.4 mostrando de forma mais detalhada as dependências dos componentes em relação as suas respectivas interfaces.

À medida que as seções a seguir se aprofundam nos detalhes de cada componente, os diagramas de classes apresentam todas as interfaces expostas e requeridas por eles.

4.2.3 SUPORTE

O componente de suporte é responsável pela iniciação, aquisição/conversão de dados analógicos (temperaturas) e atuação de comandos ON/OFF do SWPDC. A Figura 4.7 mostra um diagrama com as classes e interfaces de suporte.

O componente **Suporte** fornece as seguintes interfaces:

- **IIniciadorHW**: iniciação do sistema de software e hardware (*boot*).
- **IAtuadorHW**: incorporação dos mecanismos de atuação no hardware.
- **ISujeitoEventos**: sujeito a notificar ocorrência de eventos.
- **ITemporizador**: funcionalidade de geração do relógio interno do PDC.

A classe **pdcsys** representa as definições do sistema como tipos renomeados, definições de constantes de limites, inclusão de dependências com bibliotecas externas. Esta classe é pública em todo o contexto do SWPDC.

As classes definidas no componente Suporte são as seguintes:

- **MonitorErrosHW**: tarefa de regime permanente para monitoramento de erros de hardware, notificador de eventos.
- **AtuadorHW**: realiza a interface IAtuadorHW, conjunto de funcionalidades para atuação com o hardware (registros especiais, I/O, etc).
- **IniciadorHW**: realiza a interface IIniciadorHW, conjunto de funcionalidades para apoio a iniciação do hardware.
- **Temporizador**: ativado por interrupção externa e mantém contadores de 32 bits com resolução de 1 milissegundo; dentre eles, o relógio principal do PDC.

Bibliotecas externas:

- **uPDS3300**: declarações e protótipos para acesso às interfaces físicas do hardware. Refere-se ao modelo de engenharia adotado.
- **KR-51 RTOS**: declarações e protótipos para acesso a funcionalidades do *kernel* RTOS KR-51, provido pelo ambiente de desenvolvimento.

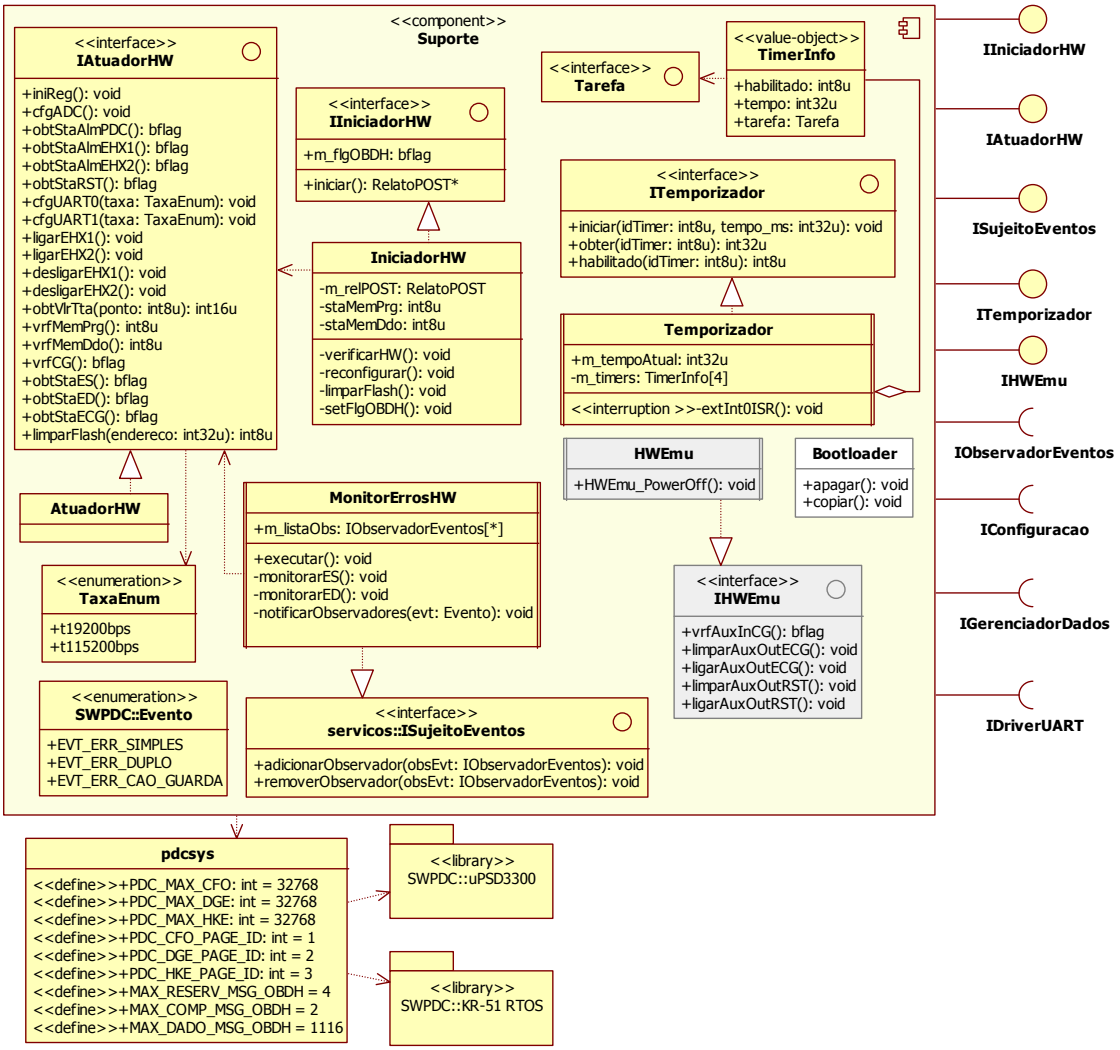

Figura 4.7: Suporte [CLS-SUP-01].

A classe **pdcsys**, da forma como foi projetada, fornece algum suporte à portabilidade do sistema em relação à plataforma de hardware. Uma mudança na plataforma de hardware implicaria em mudanças isoladas na configuração de dependência de bibliotecas externas e descrição do hardware.

4.2.4 COMUNICAÇÃO

Esta seção descreve o componente responsável pela comunicação do SWPDC com o OBDH e com os EPPs.

4.2.4.1 OBDH

A Figura 4.8 contém um diagrama classe que revela classes e interfaces no contexto do componente de **Comunicacao**, para implementação do protocolo PDC-OBDH.

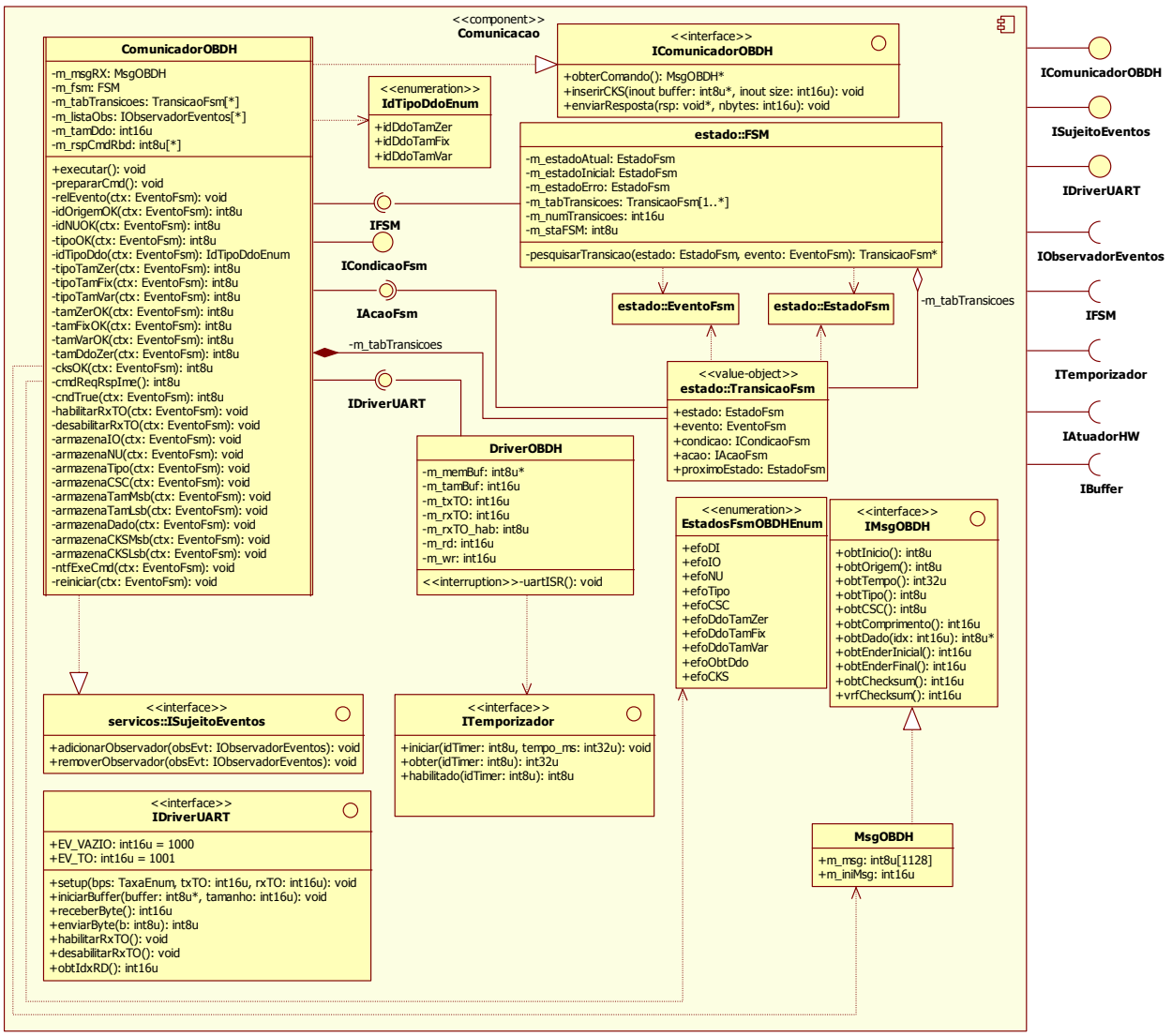

Figura 4.8: Comunicação - PDC-OBDH [CLS-COM-01].

O núcleo deste contexto é a classe ativa **ComunicadorOBDH,** que realiza as interfaces **IObservadorEventos** e **IComunicadorOBDH.**

A análise de comandos que chegam byte-a-byte a partir da porta serial (**DriverOBDH**) é apoiada pelas classes **FSM**, **EventoFsm**, **EstadoFsm** e **TransicaoFsm**. Juntas, elas definem um mecanismo de máquina de estados finitos estendida, que são ligadas ao **ComunicadorOBDH** pelas interfaces **IFSM**, **ICondicaoFsm** e **IAcaoFsm**. A relação entre **ComunicadorOBDH** e **FSM** fica a cargo da tabela de transição agregada pelo **ComunicadorOBDH** e referenciada pela classe **FSM**.

A interface **IDriverUART** abstrai a complexidade de manipulação das portas serias e, neste contexto, é realizada pela classe **DriverOBDH**.

Classes participantes:

- **ComunicadorOBDH**: tarefa de regime permanente que realiza a comunicação PDC-OBDH.
- **DriverOBDH**: porta de comunicação com a UART1, designada para a comunicação com OBDH.
- **TransicaoFsm**: tabela de transição para FSM.
- **EventoFsm**: tipo ordinal para representar eventos FSM genéricos.
- **EstadoFsm**: tipo ordinal para representar estados FSM genéricos.
- **MsgOBDH**: representação da estrutura de mensagens do protocolo PDC-OBDH.

Interfaces:

- **ISujeitoEventos**: sujeito a notificar ocorrência de eventos.
- **IDriverUART**: operações de tratamento de porta serial.
- **IObservadorEventos**: interface requerida para que este contexto seja notificado quando da ocorrência de eventos.
- **IComunicadorOBDH**: operações que implementam o protocolo de comunicação PDC-OBDH, realizada pela classe ativa **ComunicadorOBDH**.
- **IFSM**: operações sobre máquinas de estados finitos estendida.
- **ICondicaoFsm**: verificação de condições para transição de estado da FSM.
- **IAcaoFsm**: execução de ação (função) na transição de estado da FSM.
- **ITemporizador**: viabiliza a contagem de tempo para determinar o *timeout* na análise de mensagens.

Enumerações:

- **EstadosFsmOBDHEnum**: estados possíveis durante a análise de comandos OBDH.
- **IdTipoDdoEnum**: tipos de dados que podem existir numa mensagem de comandos OBDH.

4.2.4.2 EPP

A Figura 4.9 contém um diagrama classe que revela classes e interfaces no contexto do componente de **Comunicacao**, para a implementação do protocolo PDC-EPP.

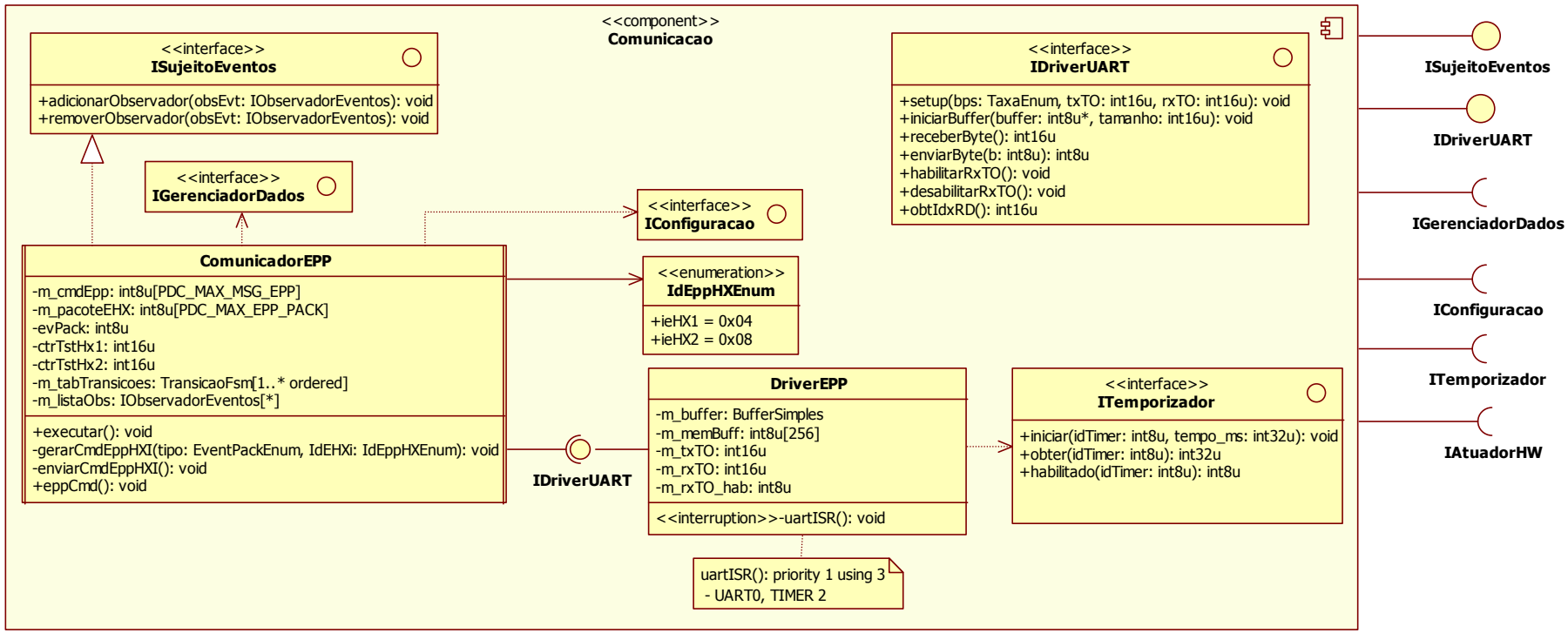

Figura 4.9: Comunicação - PDC-EPP [CLS-COM-02].

O núcleo deste contexto é formado pela classe ativa **ComunicadorEPP,** que realiza as interfaces **ISujeitoEventos** e é responsável pelo *pulling* de disparo de comandos para os conjuntos EPP-HX1 e EPP-HX2 (EPPs).

A classe **DriverEPP** é responsável pelo envio de comandos e recepção de dados aos/dos EPPs. A interface **IDriverUART,** abstrai a complexidade de manipulação das portas serias e, neste contexto, é realizada pela classe **DriverEPP**.

Classes participantes:

- **ComunicadorEPP**: tarefa de regime permanente que realiza a captura das respostas de comandos PDC-EPP, e gera os comandos a serem enviados aos EPPs.
- **DriverEPP**: comunicação com a UART0, designada para a comunicação com EPP.

Interfaces:

- **ISujeitoEventos**: sujeito a notificar ocorrência de eventos.
- **IDriverUART**: operações de tratamento de porta serial.
- **IConfiguracao**: operações de acesso aos dados de configuração do sistema.
- **ITemporizador**: viabiliza a contagem de tempo para determinar o *timeout* na análise de mensagens.
- **IConfiguracao**: para acesso à memória de configuração do SWPDC; neste contexto é usada para decidir qual será o tipo de dado (científico/diagnóstico/teste) deve ser pedido aos EPPs.

Enumerações:

- **IdEppHXEnum**: identificador dos conjuntos EPPs (EPP H1 - HX1 e EPP H2 -HX2).

4.2.5 DADOS

A Figura 4.10 ilustra o diagrama classes e interfaces no contexto do componente **Dados**, responsável pela persistência dos dados adquiridos ou gerados pelo SWPDC, além de definir estruturas de manipulação de *buffers* paginados e não paginados, de acordo com a arquitetura de hardware definida em (AD5).

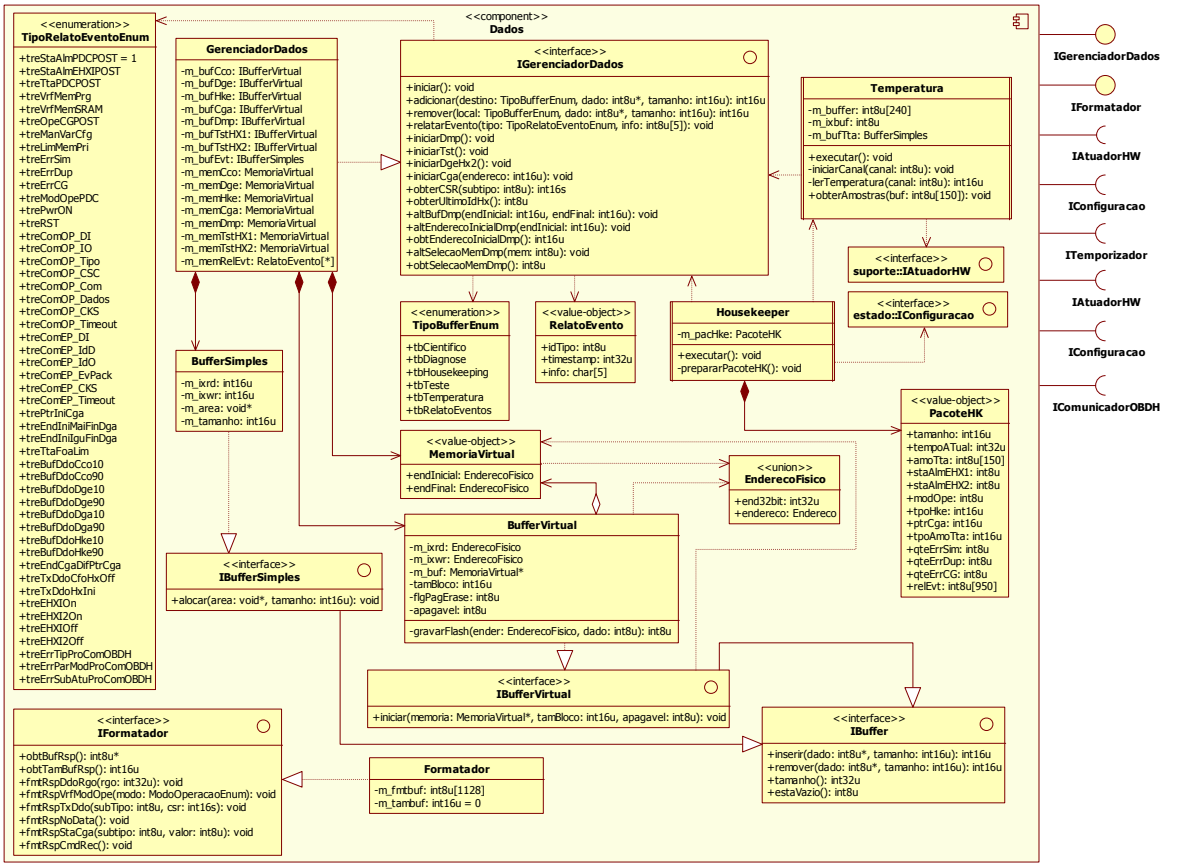

Figura 4.10: Gerenciador de Dados [CLS-DDO-01].

A parte central do componente **Dados** é formado pelas classes **GerenciadorDados**, **Housekeeper** e **Temperatura**. A classe **GerenciadorDados**, como seu próprio nome indica, é responsável pela persistência dos dados do SWPDC, em *buffers* de categorias distintas para cada tipo de dado. A interface **IBuffer**, abstrai as operações sobre *buffers* de qualquer natureza. Entretanto, as interfaces **IBufferVirtual** e **IBufferSimples** especializam **IBuffer** para representar dados que se propagam por múltiplas páginas e aqueles que residem na memória comum a todas as páginas, respectivamente.

A tarefa responsável pela geração periódica de dados de *housekeeping* é representada pela classe ativa **Housekeeper**, que tem acesso irrestrito aos membros privados de **GerenciadorDados** e **Temperatura**, onde ela busca seus dados. Além disso, outra fonte de dados de *housekeeping* é adquirida pela da interface **IConfiguracao**.

A interface **IFormatador** juntamente com a classe que a implementa – **Formatador** – oferecem serviços de formatação de dados, transformando dados no formato interno do SWPDC no formato de transmissão para o OBDH.

Gerir *buffers* é responsabilidade de **GerencidadorDados**, que delega atividades para **BufferSimples** e **BufferVirtual**, em função do tipo de dado a ser manipulado.

Classes participantes:

- **GerenciadorDados**: manipulador dos dados do SWPDC.

- **BufferSimples**: *buffer* circular para dados que residem na memória comum a todas as páginas, usando a memória SRAM (*rs0*), de acordo com o mapeamento apresentado.
- **BufferVirtual**: *buffer* circular para dados que permeiam múltiplas páginas na área de memória paginada, mapeada na memória FLASH principal.
- **Housekeeper**: tarefa de regime permanente e periódica geradora de dados de *housekeeping*.
- **Temperatura**: tarefa de regime permanente e periódica coletora de dados de temperatura.
- **RelatoEvento**: estrutura de dados para representar relatos de evento.
- **PacoteHK**: estrutura de dados para repensar dados coletados durante um *housekeeping*.
- **Formatador**: serviços de formatação de dados para transmissão.

Interfaces:

- **IBuffer**: operações de *buffer* de dados genéricos.
- **IBufferSimples**: operações para tratamento de *buffer* de memória não paginada.
- **IBufferVirtual**: operações para tratamento de *buffer* de memória paginada.
- **IGerenciadorDados**: interface fornecida pelo componente para acesso a dados.
- **IFormatador**: interface de acesso aos serviços de formatação de dados para transmissão.

Enumerações:

- **TipoRelatoEventoEnum**: enumeração dos tipos de relatos de eventos possíveis de serem gerados.
- **IdPaginaEnum**: identificação das máscaras de mapas de bit para marcação de páginas de dados.
- **TipoBufferEnum**: categorização dos tipos de *buffers*.

Complementando a visão do componente de gerenciamento de dados, a Figura 4.11 mostra o projeto do mapeamento da memória FLASH do modelo de engenharia do SWPDC.

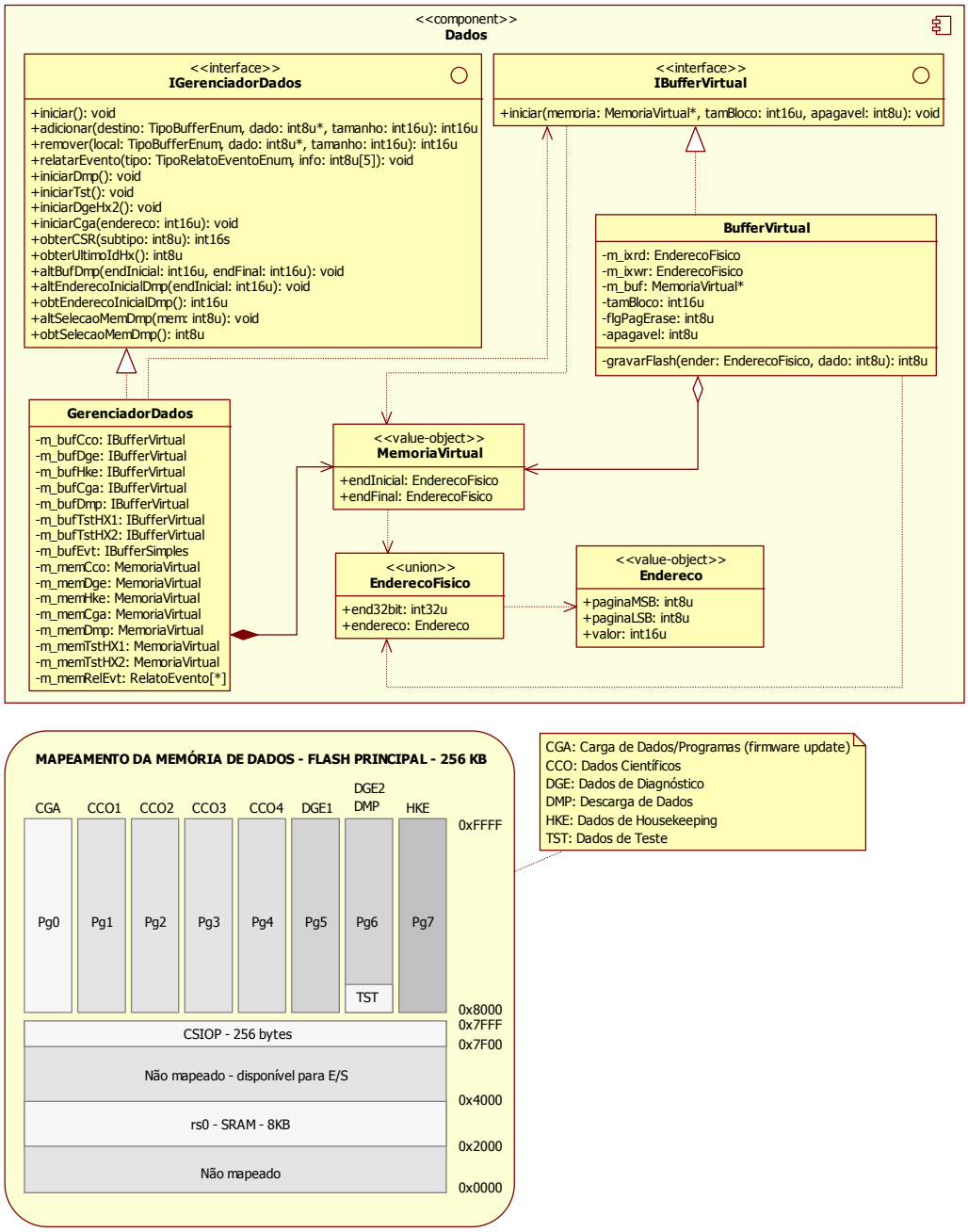

Figura 4.11: Memória Virtual [CLS-DD0-02].

A página 0 (zero) está destinada para carga de programas. As páginas de 1 a 4 são reservadas para persistir dados científicos. Dados de diagnóstico ficarão nas páginas 5 e 6, com a página 6 compartilhada com os dados de descarga (*dump*) e dados de teste. Assim, dados de descarga, dados de diagnósticos e dados de teste pode se sobrepor mutuamente. A página 7 está dedicada aos dados de housekeeping.

4.2.6 ESTADO

O gerenciamento de estado é definido do componente **Estado**. Entende-se por estado, neste contexto, o conjunto formado pelas entidades sensíveis à dinâmica de operação do SWPDC, como por exemplo, dados de configuração, tabelas de regras de permissão de execução de comandos, monitoramento de cão-de-guarda e o mecanismo de máquinas de estados finitos estendida.

A Figura 4.12 mostra o diagrama de classes do componente Estado.

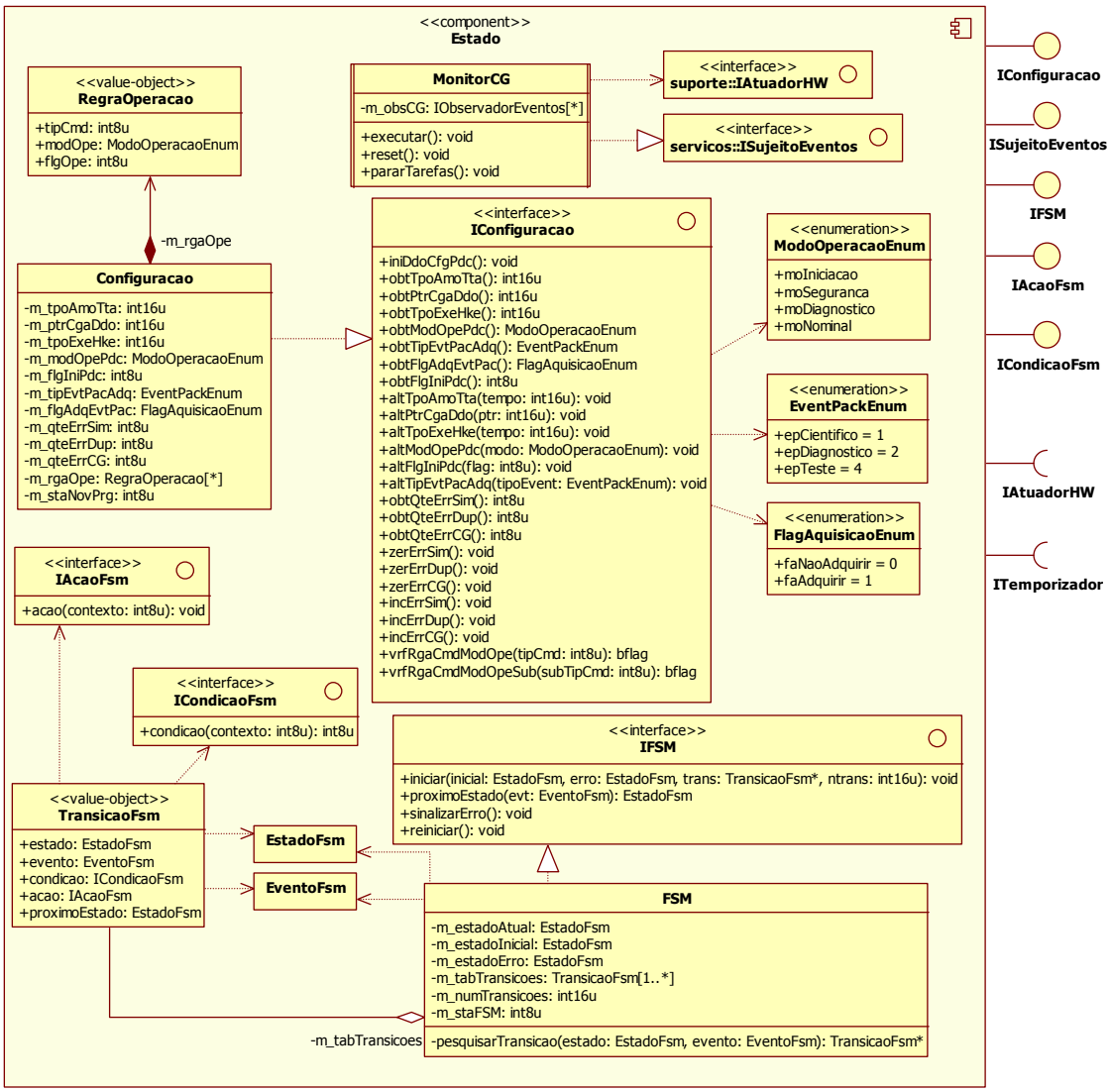

Figura 4.12: Componente Estado [CLS-EST-01].

Classes participantes:

- **Configuracao**: realiza a interface **IConfiguracao** e detém os dados de configuração do SWPDC.

- **TransicaoFsm**: tabela de transição para FSM.
- **EventoFsm**: tipo ordinal para representar eventos FSM genéricos.
- **EstadoFsm**: tipo ordinal para representar estados FSM.
- **MonitorCG**: tarefa de regime permanente que manipula o cão-de-guarda e notifica erros gerados por este dispositivo.
- **RegraOperacao**: determina a relação entre tipo de comando do OBDH e modo de operação, indicando se um comando pode ser executado num dado modo de operação.
- **FSM**: realiza a interface **IFSM**, criando a base do mecanismo de máquina de estados finitos estendida.

Interfaces:

- **IConfiguracao**: operações de manipulação de dados de configuração.
- **IAcaoFsm**: operação de ação de uma FSM a executar na ativação de uma transição.
- **ICondicaoFsm**: operação de verificação de condição para ativar transições de FSM.
- **IFSM**: operações de manipulação de máquina de estados finitos estendida
- **ISujeitoEventos**: interface requerida para que este contexto notifique a ocorrência de eventos.
- **IAtuadorHW**: operações de acesso e atuação no hardware, providas pelo componente de suporte.

Enumerações:

- **FlagAquisicaoEnum**: indicares de aquisição e não aquisição de dados do EPP.
- **EventPackEnum:** indicadores do tipo de dado a adquirir por meio do EPP.
- **ModoOperacaoEnum**: modos de operação do SWPDC.

4.2.7 SERVIÇOS

O componente de serviços é responsável pela determinação da execução de comandos e concentra as classes que implementam a execução de cada comando previsto pelo protocolo de comunicação PDC-OBDH (AD6).

Uma particularidade que pode ser notada no componente de serviço é que todo o sistema converge para ele no sentido das dependências das interfaces oferecidas pelos demais componentes do sistema.

Nesse componente reside a classe **SWPDC** que fornece o primeiro fluxo de controle na iniciação do software. Além dela, também são definidas as estruturas de manipulação de comandos e observadores de eventos da aplicação.

A Figura 4.13 apresenta o diagrama de classes que expõe sua organização estática.

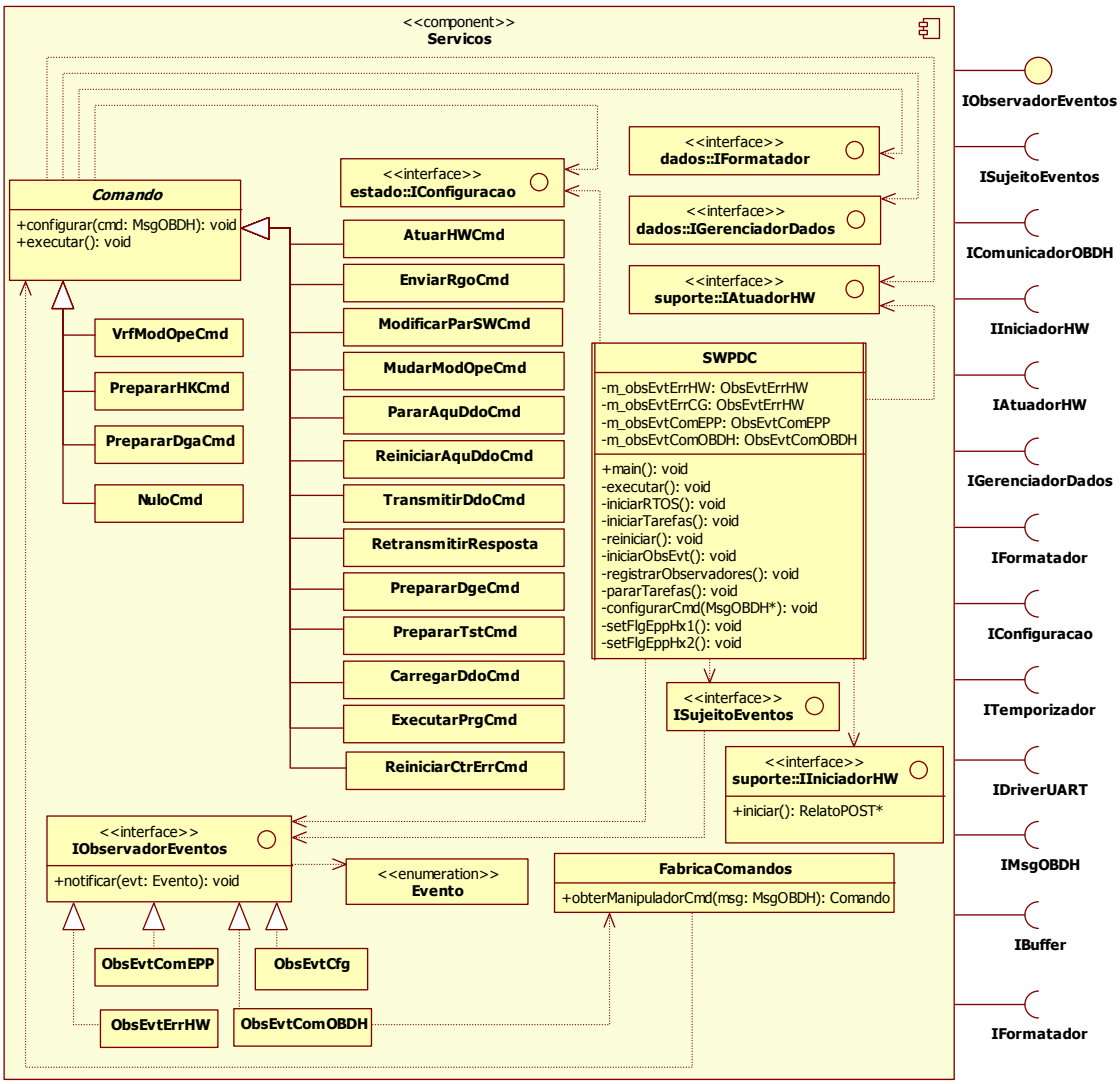

Figura 4.13: Serviços [CLS-SVC-01].

Classes participantes:

- **Comando**: classe abstrata para referenciar manipuladores de comando.
- **FabricaComandos:** devolve referências a manipuladores de comando em função de uma mensagem de comando do OBDH.
- **SWPDC**: classe detentora do fluxo de controle principal. Possui referências para observadores de eventos e manipuladores de comandos.
- **ObsEvtErrHW**: observador de eventos de erros de hardware.
- **ObsEvtComOBDH**: observador de eventos provenientes da comunicação com o OBDH.
- **ObsEvtComEPP**: observador de eventos provenientes da comunicação com os conjuntos EPP-HXI.
- **ObsEvtCfg**: observador de ocorrências de mudanças nos dados de configuração, principalmente interessado na mudança de modos de operação.

- **NuloCmd**: comando *dummy* devolvido pela **FabricaComandos** caso não encontre um tipo de comando específico. Faz parte do isolamento de erros, pois neste contexto uma mensagem de comando está sintaticamente correta e sempre deverá ter um manipulador de comando associado. Se uma instância da classe **NuloCmd** for referenciada nada ocorrerá, pois suas operações têm implementações vazias.
- **AtuarHWCmd**: comando de atuação no hardware.
- **EnviarRgoCmd**: comando para obtenção e envio de dados de relógio.
- **ModificarParSWCmd**: comando para modificação de parâmetros de software.
- **MudarModOpeCmd**: comando para chavear de modo de operação.
- **PararAquDdoCmd**: comando para interromper a aquisição de dados em andamento.
- **ReiniciarAquDdoCmd**: comando para retomar a aquisição de dados científicos.
- **TransmitirDdoCmd**: comando para executar a transferência de dados requisitados do PDC para o OBDH.
- **RetransmitirRspCmd**: comando para retransmitir a última resposta da transferência de dados da sessão corrente.
- **PrepararDgeCmd**: comando para iniciar uma sessão de aquisição e transferência de dados de diagnóstico.
- **PrepararDgaCmd**: comando para iniciar uma sessão de descarga de dados.
- **PrepararTstCmd**: comando para iniciar uma sessão de aquisição e transferência de dados de teste.
- **PrepararHKCmd**: comando para iniciar uma sessão de *housekeeping*.
- **CarregarDdoCmd**: comando para carregar dados/programas na memória.
- **ExecutarPrgCmd**: comando para executar o programa carregado previamente.
- **ReiniciarCtrErrCmd**: comando para zerar os contadores de erro.

Interfaces:

- **IObservadorEventos**: interface realizada pelos observadores de evento para que este contexto seja notificado quando da ocorrência de eventos nos diversos componentes.
- **ISujeitoEventos**: interface requerida para que este contexto seja notificado quando da ocorrência de eventos.
- **IComunicadorOBDH**: operações que implementam o protocolo de comunicação PDC-OBDH, realizada pela classe ativa **comunicacao::ComunicadorOBDH**.
- **IConfiguracao**: operações de manipulação de dados de configuração.
- **IAtuadorHW**: operações de acesso e atuação no hardware, providas pelo componente de suporte.
- **IIniciadorHW**: iniciação do sistema de software e hardware (*boot*), oferecida pelo componente de suporte.
- **IFormatador**: interface do componente de dados para acesso à funcionalidade de formatação de dados para transmissão.
- **IGerenciadorDados**: interface fornecida pelo componente de acesso a dados para obtenção dos diversos tipos de dados que podem ser adquiridos ou gerados pelo sistema.

Enumerações:

- Não há**.**

Um outro serviço do SWPDC é a atualização de *firmware* que pode ser eventualmente comandada pelo OBDH por meio de sucessivas cargas de pacotes de programas. O mapeamento

da memória virtual para a realização das atividades de carga, verificação e substituição do novo *firmware* está descrito na Figura 4.14. A configuração da memória virtual dar-se-á pela manipulação do registrador VM do CSIOP, de acordo com o definido em RD10.

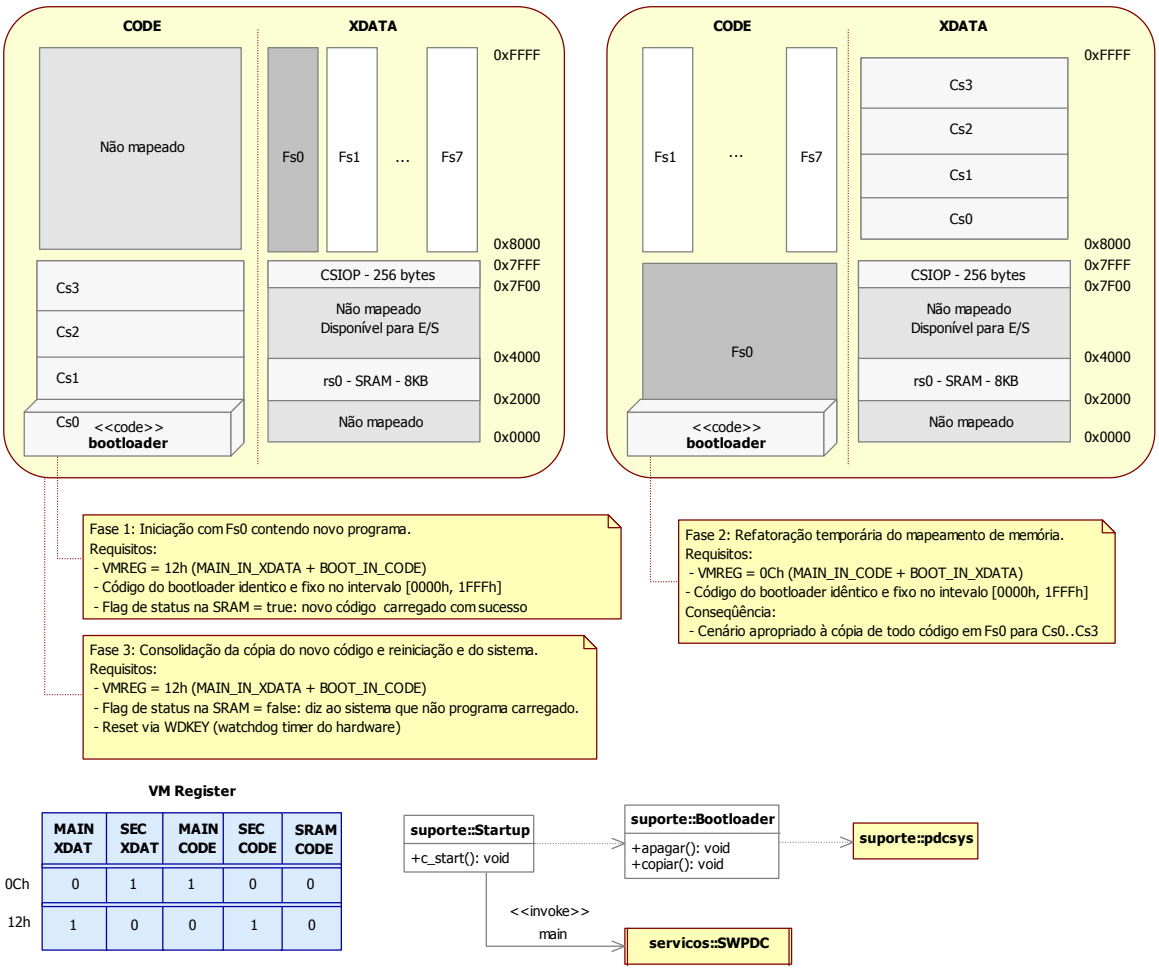

Figura 4.14: Configuração de memória virtual para substituição de *firmware*.

4.3 ARQUITETURA DINÂMICA

A Figura 4.15 apresenta uma visão dinâmica dos componentes do software SWPDC descritos na seção 4.2. Na arquitetura dinâmica, aspectos de interação e dependência entre os componentes são ressaltados.

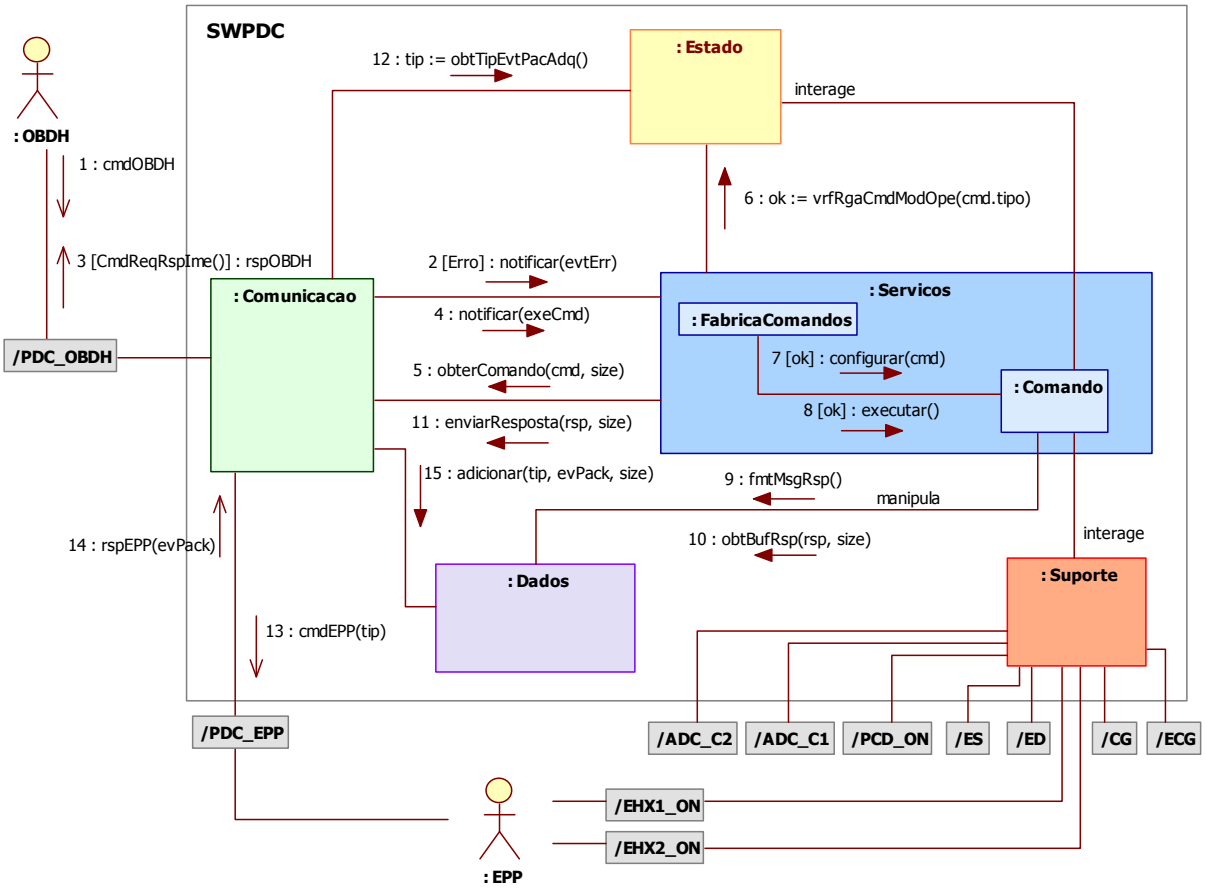

Figura 4.15: Arquitetura Dinâmica do Software PDC.

As seções a seguir descrevem os modelos comportamentais dos componentes especificados na seção 4 usando diagramas UML.

De forma geral, é possível verificar pela Figura 4.15 a interatividade entre os componentes do sistema. Além disso, essa visão representa o pleno funcionamento do SWPDC, com interações síncronas e assíncronas entre as entidades internas e externas, além da atuação no hardware através das portas mapeadas na memória de I/O e canais analógicos, recebimento de comandos do OBDH, obtenção, formatação e envio de dados em mensagens resposta. Exceto pela fase de iniciação do SWPDC, todo o sistema funciona como descrito nessa visão geral da arquitetura dinâmica, com particularidades inerentes à determinação e execução de comandos do

OBDH, em função do modo de operação corrente e outros dados da configuração vigente no sistema.

4.3.1 INICIAÇÃO DO SWPDC – FASE 1

A iniciação do SWPDC foi dividida em duas fases. A primeira, descrita nesta seção, refere-se à verificação do hardware (POST) e a segunda modela o que acontece após o POST.

O diagrama da Figura 4.16 ilustra a seqüência de operações para realização do *boot* do SWPDC.

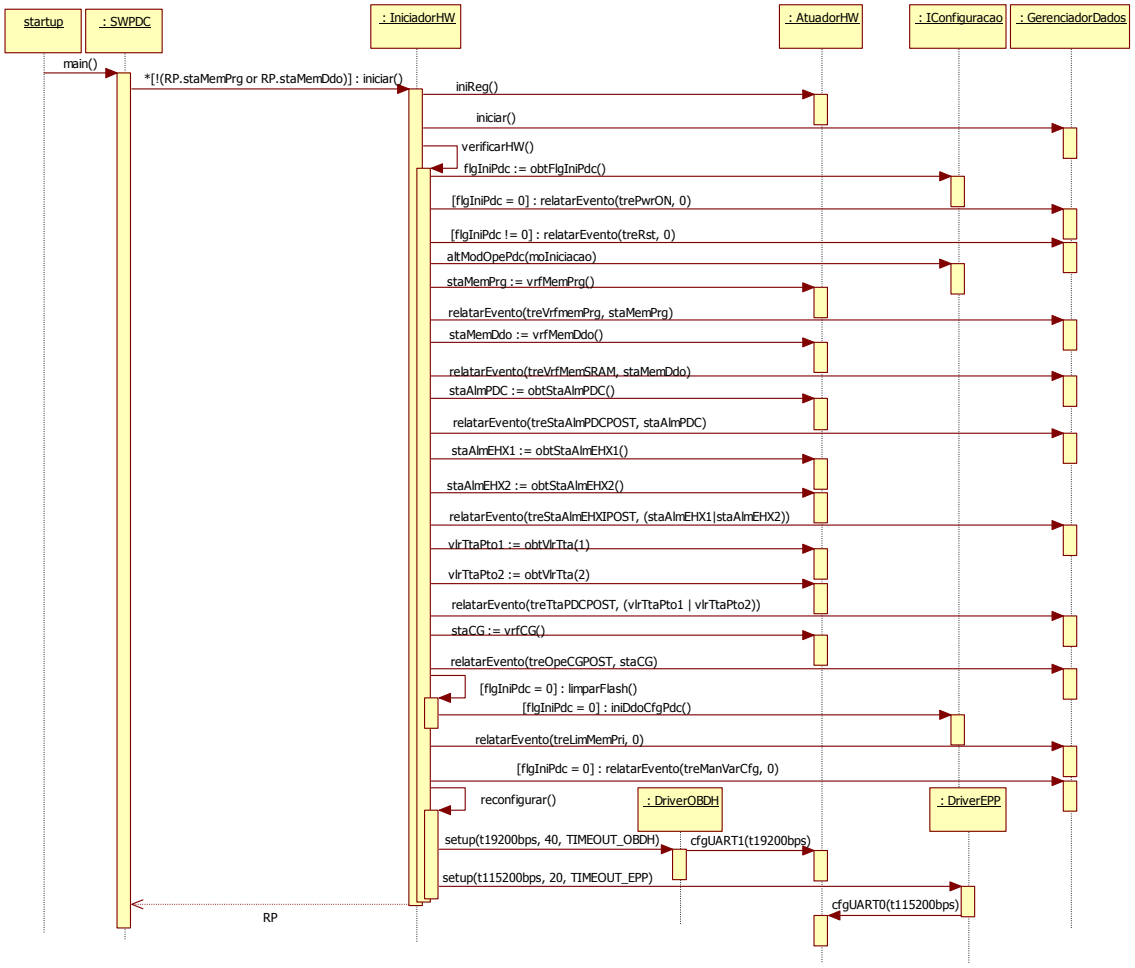

Figura 4.16: Iniciação do SWPDC - Fase 1 [ITE-INI-01].

Quando a rotina de ***startup*** inicia ela chama obrigatoriamente a função de entrada principal (***main()***) do SWPDC, que fica em *loop* até que a memória de programa e a memória de dados sejam iniciadas com sucesso.

Durante este processo, ocorre a verificação do hardware efetuada pelo iniciador do hardware (**IniciadorHW**).

Em seguida, os dados de verificação são adquiridos e imediatamente armazenados na forma de relato de eventos no gerenciador de dados.

4.3.2 INICIAÇÃO DO SWPDC – FASE 2

A segunda fase da iniciação do SWPDC ocorre imediatamente após uma verificação bem sucedia do hardware com os relatos de POST. A Figura 4.17 ilustra o processo de término do POST, com a saída do modo de iniciação, entrando no modo de segurança, ativando instâncias das classes ativas do sistema.

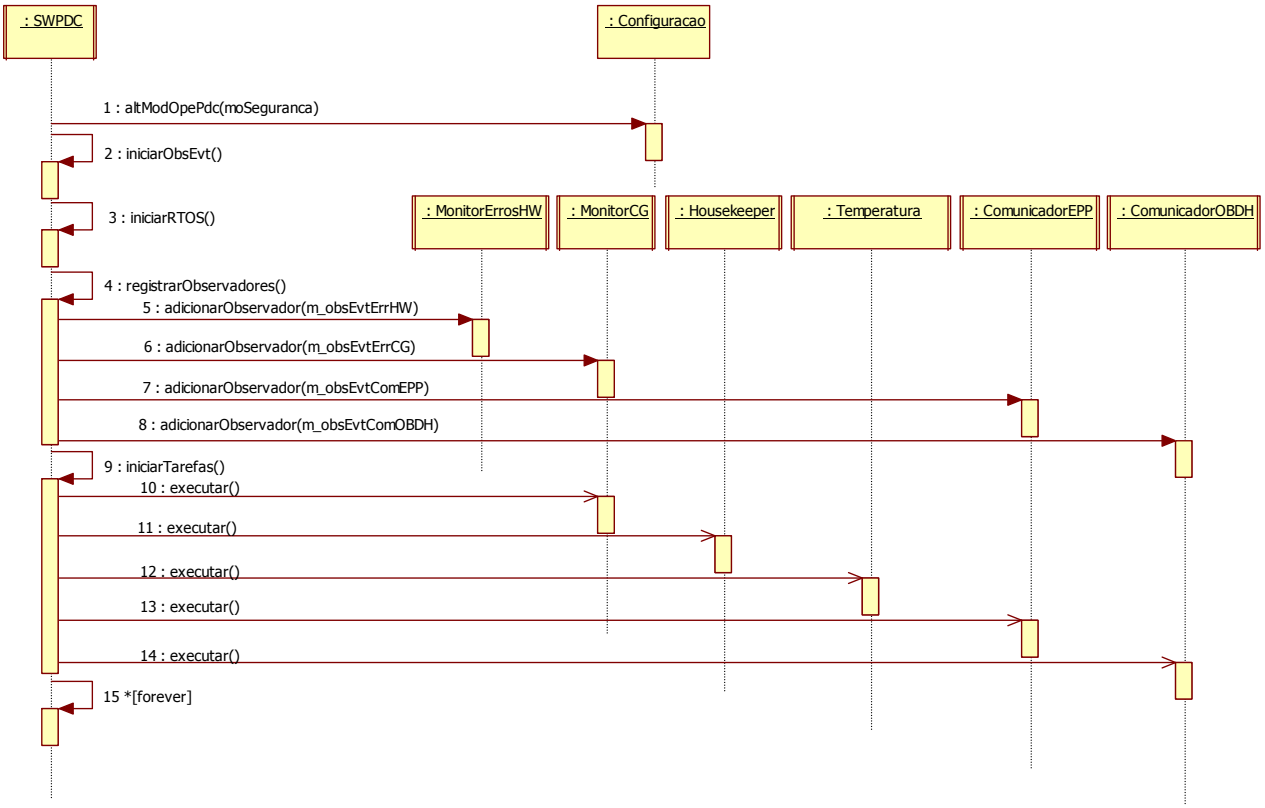

Figura 4.17: Iniciação do SWPDC - Fase 2 [ITE-INI-02].

As atividades realizadas neste contexto são:

- Iniciação dos observadores de eventos do sistema.
- Iniciação das rotinas do sistema operacional.
- Registro dos observadores de eventos nos respectivos sujeitos (notificadores).
- Iniciação das tarefas de regime permanente (classes ativas).

Ao final da iniciação, a linha de execução principal do SWPDC entra num *loop* infinito, deixando que as classes ativas assumam o controle e cooperem entre si para o funcionamento do sistema.

4.3.3 COMUNICAÇÃO PDC-OBDH

Esta seção destaca a iniciação e o funcionamento do componente de comunicação, no contexto do tratamento de comandos enviados ao SWPDC pelo OBDH.

A primeira etapa ocorre imediatamente após a criação da tarefa **ComunicadorOBDH**, por meio do método *executar( )*. Ao ser iniciada, esta tarefa faz uma pausa de 30 segundos antes de prosseguir com a iniciação de sua máquina de estados finitos (FSM) que provê o mecanismo de análise sintática das mensagens remetidas pelo OBDH.

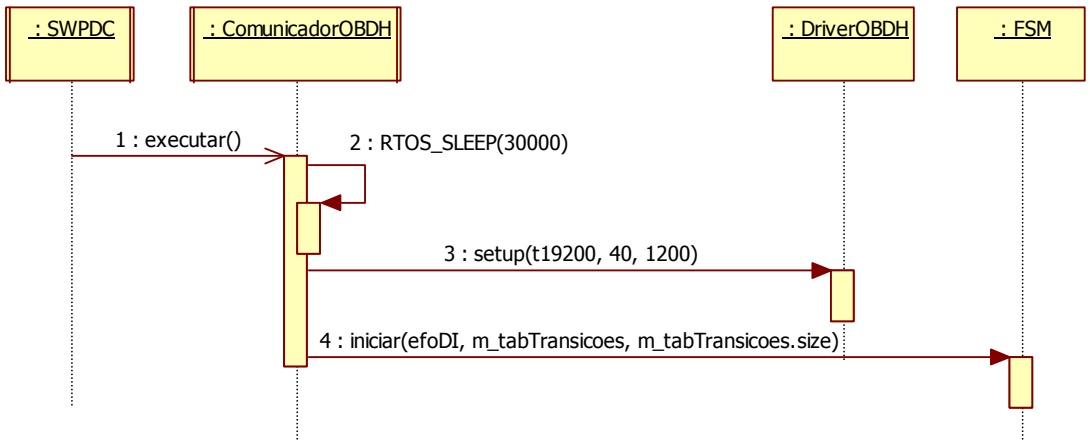

Figura 4.18: Iniciação da Comunicação PDC-OBDH [ITE-COM-01].

Após a iniciação da **FSM**, o processo segue com a configuração da porta de comunicação por meio da interface **DriverOBDH::setup()** que configura a UART designada para comunicação com OBDH, apoiada pela interface **AtuadoHW::cfgUART1()**, como ilustrado na Figura 4.18.

Com a UART configurada e pronta para ser usada, a tarefa **ComunicadorOBDH** entra em seu regime permanente, que está sujeito às interrupções da UART1, ficando assim aguardando pela chegada de bytes via este dispositivo, pela interface **DriverOBDH ::receberByte()**. Tão logo a rotina de tratamento de interrupção da UART1 detecte a chegada de um byte, ela lê e põe o byte lido em seu *buffer*, que causa a sua eventual remoção do *buffer* para alimentar a máquina de estados finitos.

Dessa forma, um byte lido da UART1 torna-se nesse contexto um evento da **FSM**, que por sua vez, verifica a correspondência entre evento versos estado atual em sua tabela de estados previamente configurada, realizando chamadas de pesquisa pelo próximo estado da **FSM** e respeitando as condições para disparar as transições de estados (Figura 4.19).

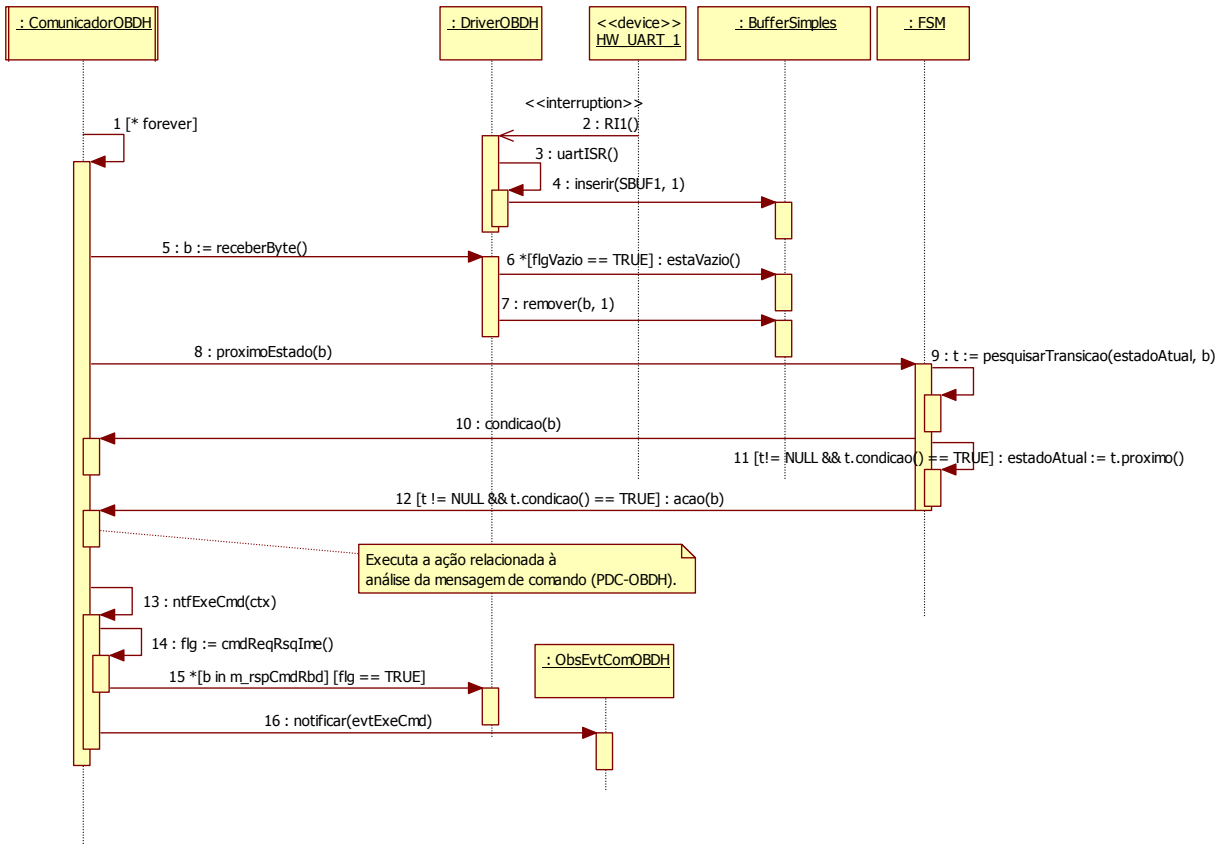

Figura 4.19: Recepção de Comandos do OBDH [ITE-COM-03].

Após a identificação da chegada de uma mensagem do protocolo PDC-OBDH sintaticamente válida, ocorre a notificação da chegada desta mensagem no componente de serviços. O **ComunicadorOBDH** determina também se a mensagem recém chegada requer resposta imediata (notificação de comando recebido). Em caso afirmativo, a mensagem COMANDO_RECEBIDO previamente configurada no membro **ComunicadorOBDH:: m_rspCmdRbd** é enviada para o OBDH.

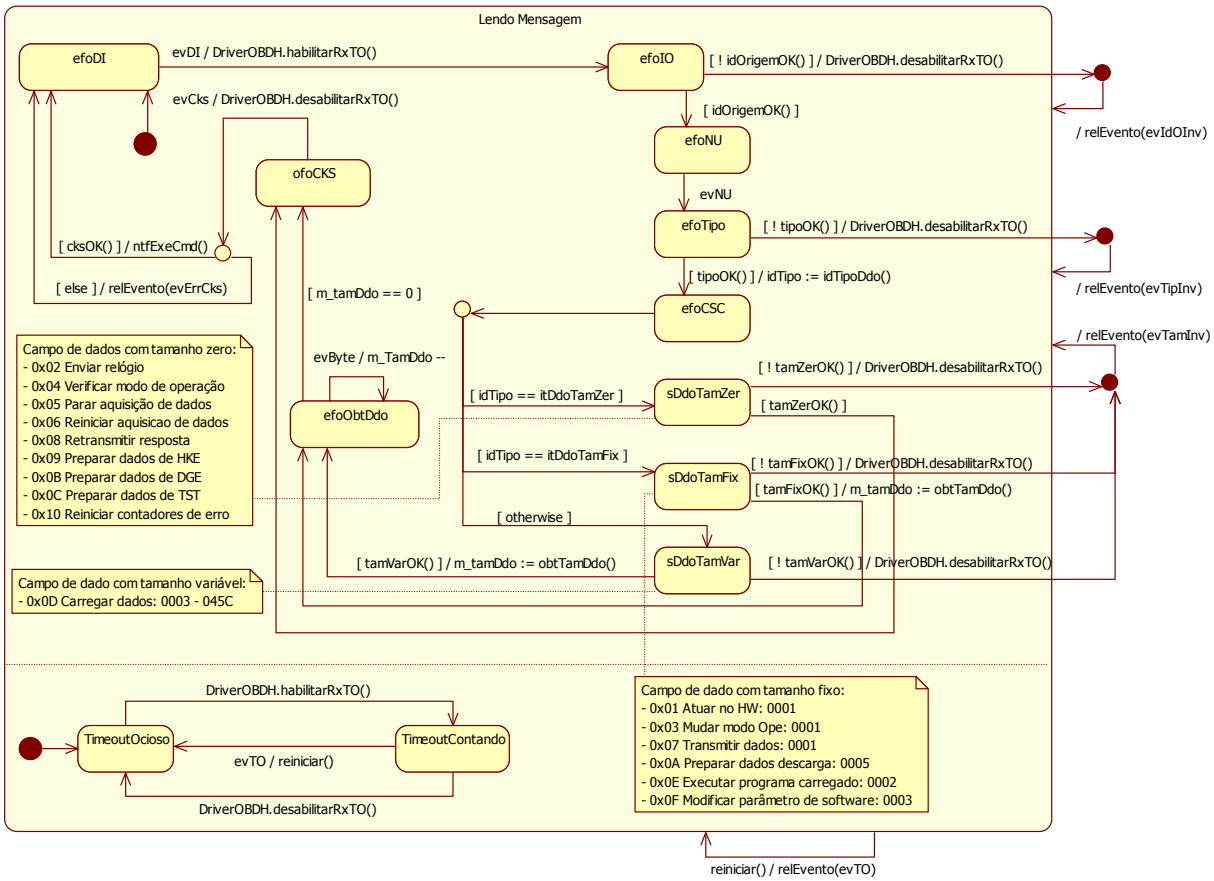

Figura 4.20: Interpretação Comando OBDH [SCH-COM-02].

O gráfico de estados (*Statecharts*) apresentado na Figura 4.20 especifica o comportamento da análise de comandos enviados pelo OBDH.

Aqui é identificado cada campo de mensagem de comando PDC-OBDH. A notificação e tratamento do *timeout* também estão previstos neste modelo.

Na ocorrência de erros durante a análise do comando, a máquina é reiniciada, o *timeout* é desligado (invalidado) e um relato de evento é gerado.

4.3.4 COMUNICAÇÃO PDC-EPP

Esta seção descreve os modelos comportamentais do componente **Comunicacao** no contexto da comunicação PDC-EPP.

O diagrama de seqüência ilustrado na Figura 4.21 modela a ativação da comunicação com EPPs. Ela inicia-se com a espera de 1 minuto, prosseguindo com a iniciação da FSM e configuração da porta UART0, de forma similar à ativação da comunicação PDC-OBDH descrita anteriormente.

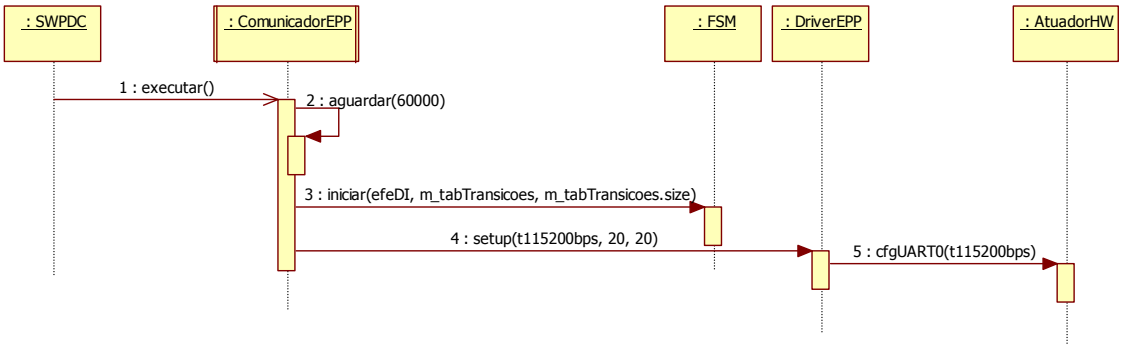

Figura 4.21: Iniciação da Comunicação PDC-OBDH [ITE-COM-01].

Com a UART configurada e pronta para ser usada, a tarefa **ComunicadorEPP** entra em seu regime permanente, que está sujeito às notificações de dado pronto da UART0, ficando, assim, aguardando pela chegada de *EventPacks* via este dispositivo Tão logo a rotina de tratamento de interrupção da UART0 detecte a presença de uma mensagem de dados integra, ocorra a notificação.

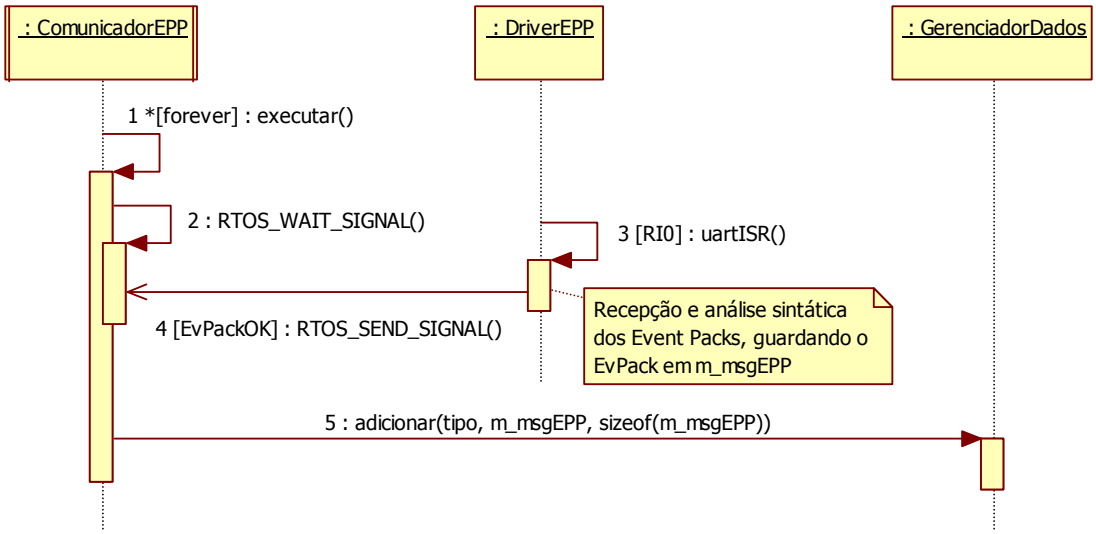

Figura 4.22: Aquisição de Telemetria [ITE-COM-04]

O diagrama mostrado na Figura 4.24 retrata os detalhes comportamentais associados com a análise dos dados provenientes dos EPPs, e faz parte da rotina de tratamento de interrupções da UART0 que, neste caso, é implementado pelo método **DriverEPP::uartISR**().

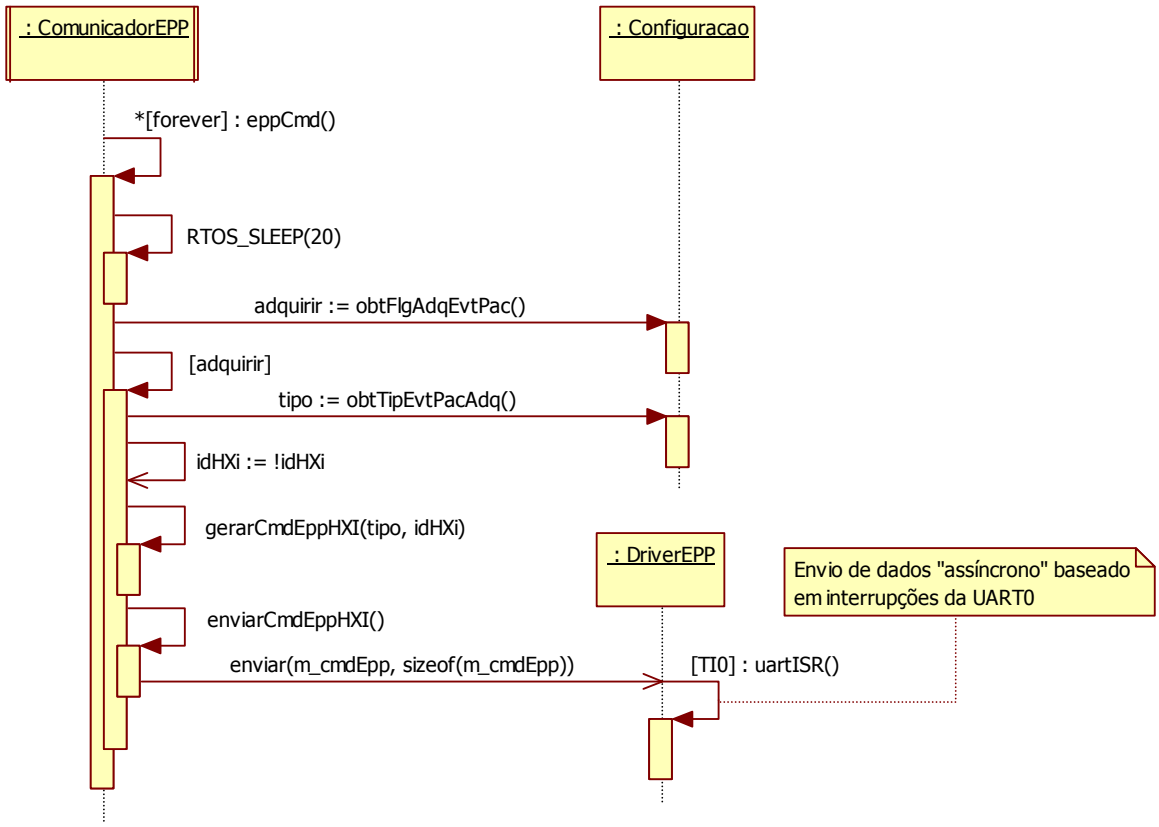

Figura 4.23: Geração de solicitações de dados aos EPPs [ITE-COM-05]

No entanto, o processo de aquisição de dos *EventPacks* (telemetria) ilustrado na Figura 4.22 não funciona sozinho. Existe um outro processo, ilustrado na Figura 4.23 que realiza o *pulling* de envio de comandos de solicitação de dados do tipo determinado pela configuração atual do SWPDC, aos EPPs.

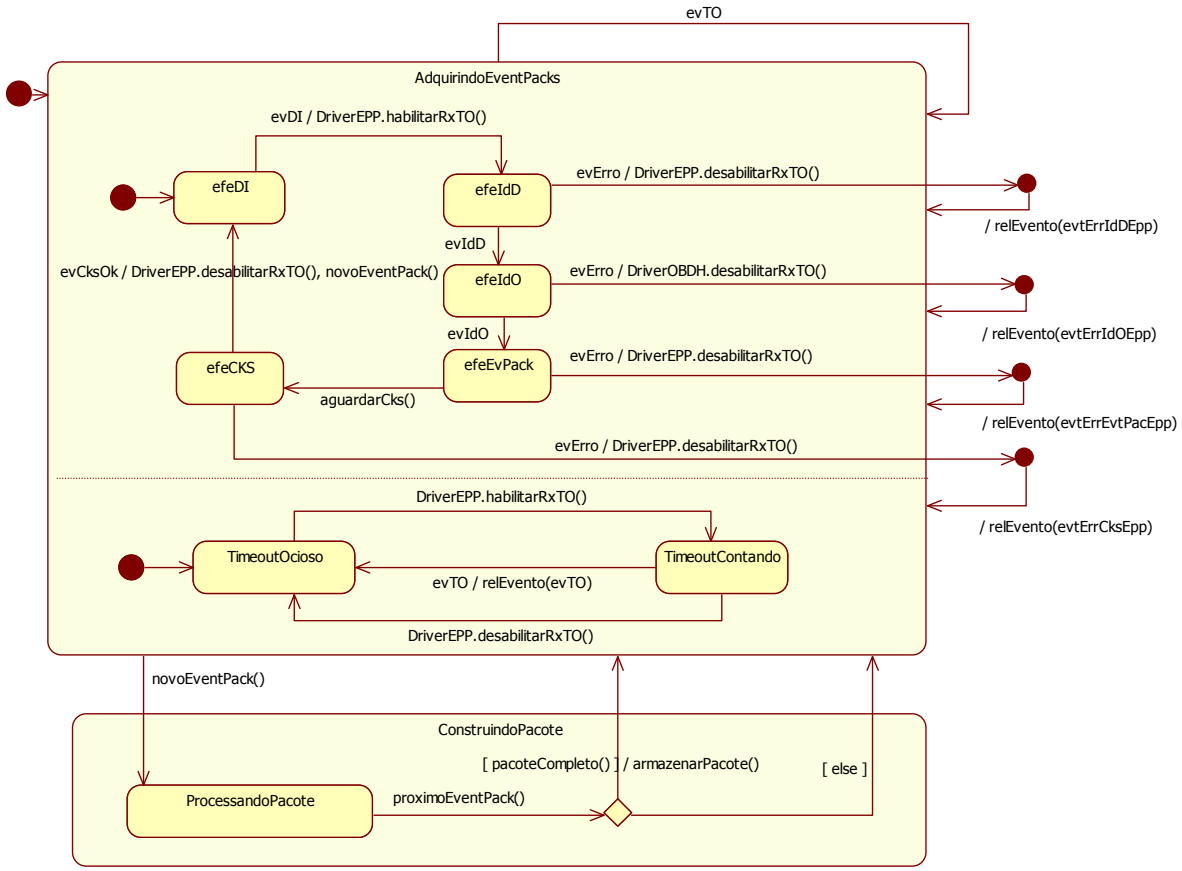

Figura 4.24: Análise de Telemetria [SCH-COM-02].

O gráfico de estados (*Statecharts*) apresentado na Figura 4.24 especifica o comportamento da análise de telemetria, ou seja, a análise dos dados que são enviados na forma de mensagens PDC-EPP em resposta aos comandos enviados durante o *pulling*.

Aqui é identificado cada campo de mensagem de comando PDC-EPP. A notificação e o tratamento do *timeout* também estão previstos neste modelo.

Na ocorrência de erros durante a análise do comando, a máquina é reiniciada, o *timeout* é desligado (invalidado) e um relato de evento é gerado.

Note que os pacotes só vão de fato para os *buffers* do gerenciador de dados quando um conjunto completo de pacotes *EventPacks* são efetivamente adquiridos de qualquer um dos conjuntos EPP-HXI.

4.3.5 AQUISIÇÃO DE DADOS DE TEMPERATURA

Esta seção detalha o comportamento da aquisição de dados de temperatura pela classe ativa **Temperatura**, no contexto do componente de gerenciamento de dados. O diagrama de seqüência da Figura 4.25 ilustra esse comportamento de regime permanente.

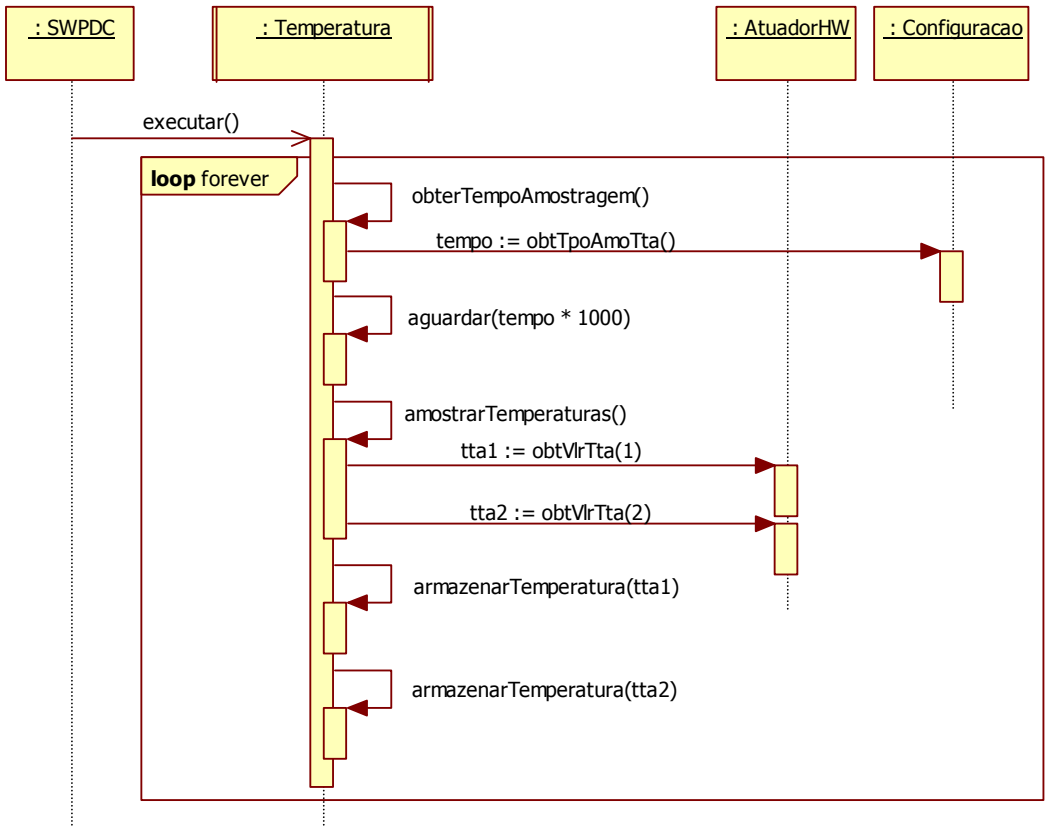

Figura 4.25: Aquisição de dados de temperatura [ITE-DDO-01].

Uma vez iniciado pelo fluxo de controle principal do SWPDC, a tarefa **Temperatura** entra em seu *loop* permanente, obtendo primeiramente o tempo de esperara para realizar a coleta das amostras e consumindo este tempo, bloqueando de acordo com o tempo especificado pela configuração do SWPDC. A multiplicação do tempo por 1000 é necessária para converter o tempo de segundo para milissegundos, requerido pela função *aguardar()*.

Ao sair do bloqueio por tempo, as temperaturas nos dois pontos determinados pela classe **AtuadorHW** são lidas e armazenadas em seu *buffer* local, no formato de 10 bits por amostra.

4.3.6 COLETA DE DADOS DE HOUSEKEEPING

A execução da tarefa de regime permanente e periódica **Housekeeper** resulta na coleta de dados de *housekeeping* que possui três fontes de dados, de acordo com a Figura 4.26:

- **Temperatura**: *buffer* de das últimas temperaturas amostradas até o momento do *housekeeping*.
- **AtuadorHW**: *status* de alimentação dos conjuntos EPP-HXI.
- **Configuracao**: informações de modo de operação, tempos, ponteiro de carga de dados e contadores de erro.

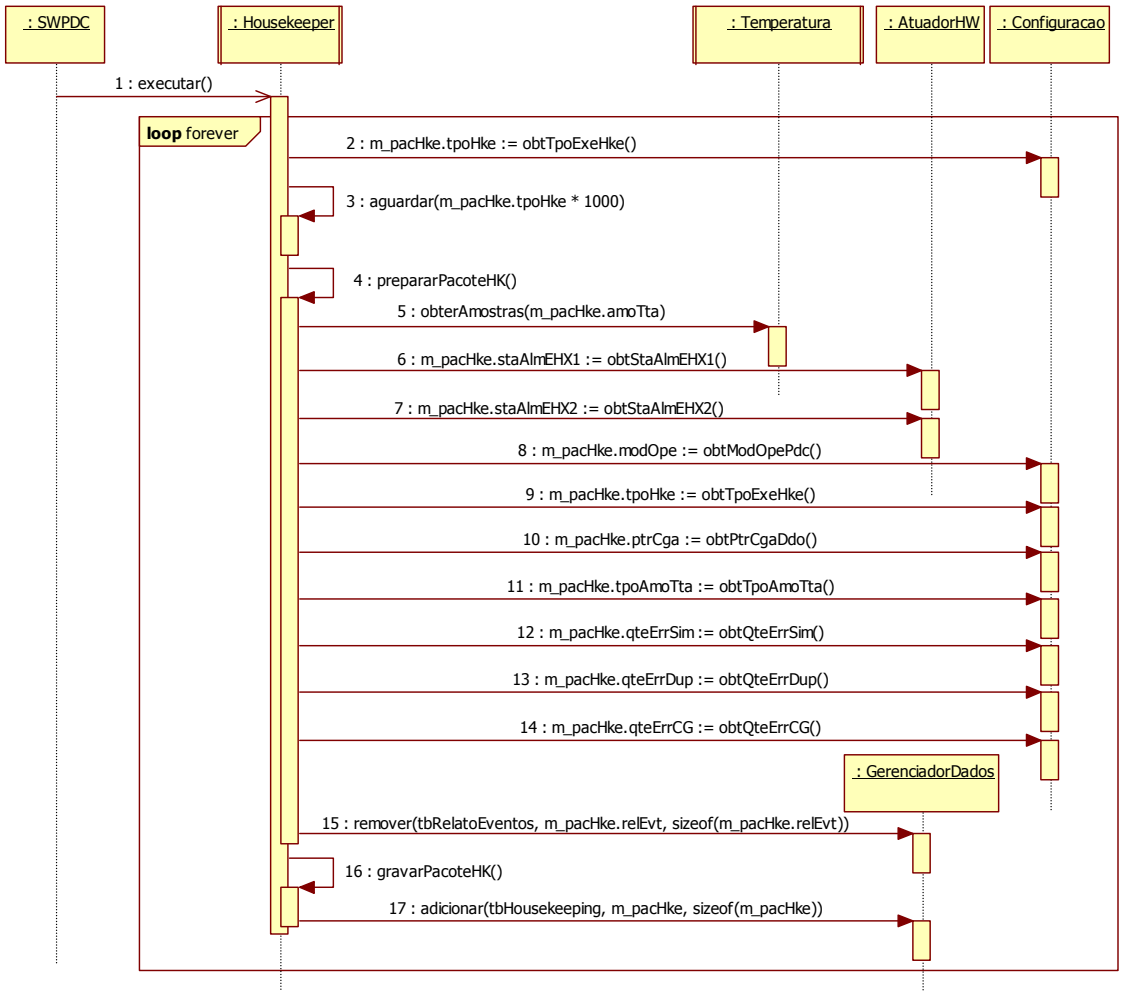

Figura 4.26: Coleta de dados de *Housekeeping* [ITE-DDO-02].

O processo se dá pelas seguintes atividades:

- Espera pelo tempo de execução da tarefa.
- Iniciação de um novo pacote de *housekeeping*.
- Aquisição de dados nas origens especificadas.
- Preenchimento do pacote de *housekeeping*

- Armazenamento do pacote de *housekeeping*.

4.3.7 MONITORAMENTO DO CÃO-DE-GUARDA

O monitoramento do cão-de-guarda é uma tarefa de regime permanente realizada pela classe ativa **MonitorCG**, de acordo com a ilustração do diagrama de seqüência na Figura 4.27.

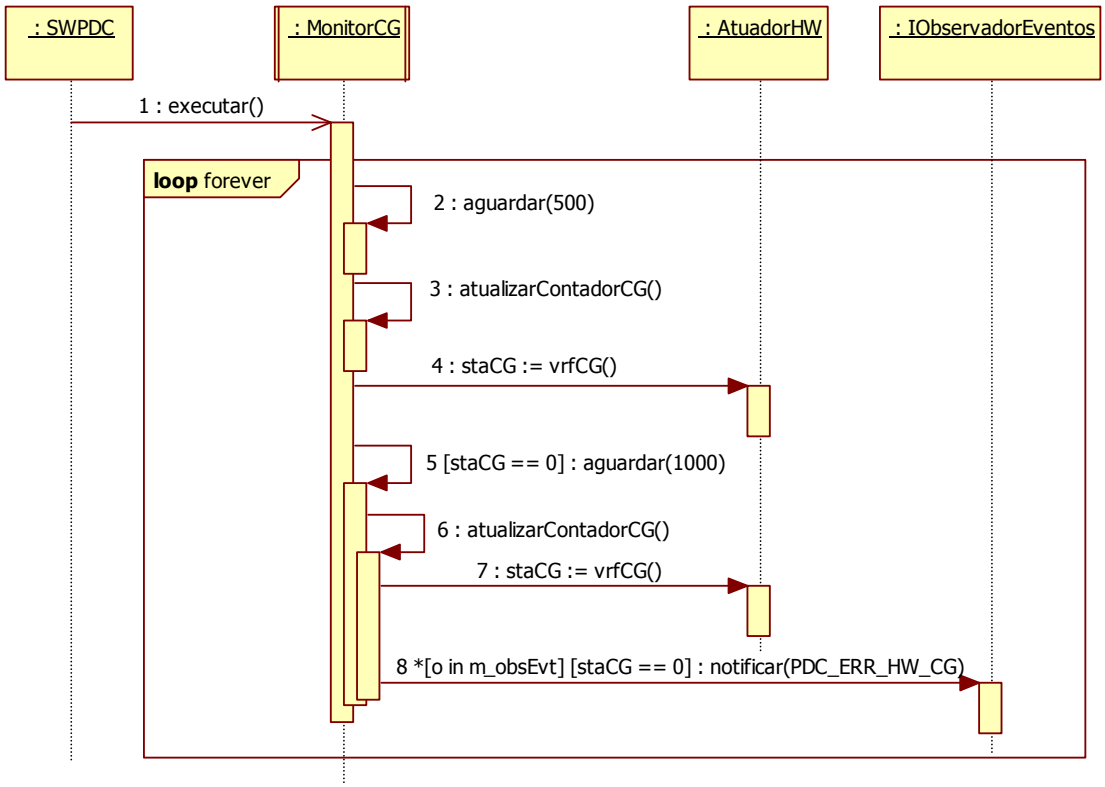

Figura 4.27: Monitoramento do Cão-de-Guarda [ITE-EST-01].

Sua responsabilidade é atualizar o contador de cão-de-guarda, sendo apoiado pela classe de atuação no hardware (**AtuadorHW**). Se o status da verificação do cão-de-guarda ao final da primeira atualização for mal sucedida, ocorre uma segunda espera com o dobro do tempo da primeira atualização, e novamente uma nova atualização e verificação são feitas. Neste caso, uma nova verificação mal sucedida resultará na notificação de um erro de hardware identificado como erro de cão-de-guarda para o observador apropriado.

4.3.8 DETERMINAÇÃO E EXECUÇÃO DE COMANDOS DO OBDH

Determinar a execução de comandos é uma das funções do SWPDC quando um novo comando proveniente do OBDH for analisado e disponibilizado para processamento pelo componente de comunicação. De acordo com o diagrama da Figura 4.28, este processo se inicia pela notificação de execução de comando para o observador de eventos de comunicação com o OBDH, que delega o processamento do comando para a classe principal **SWPDC**. De fato, a

própria função do observador (**ObsEvtComOBDH**) é uma referência para alguma função de **SWPDC** neste contexto.

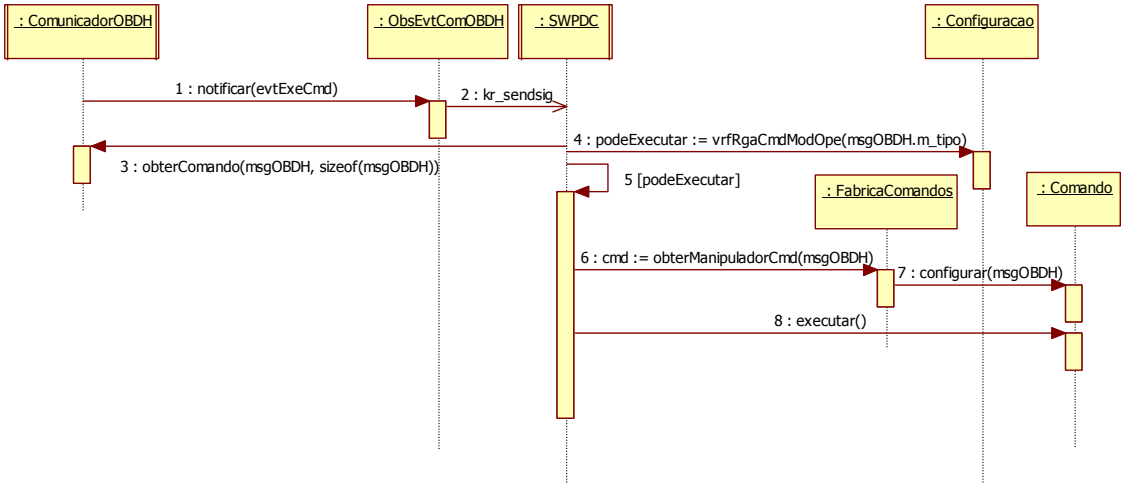

Figura 4.28: Determinação da execução de comandos OBDH [ITE-SVC-01]

Dessa forma, a determinação da execução do comando prossegue com busca da mensagem de comando por meio da interface **ComunicadorOBDH::obterComando()**. Em seguida, a interface de configuração é interrogada a respeito da permissão de execução do comando em função do modo de operação atual do sistema. Se permitido, o comando é passado para um manipulador apropriado por meio da classe **FabricaComandos**, que devolve uma referência para o mesmo.

Finalmente, o comando é executado de forma transparente para o contexto corrente, uma vez que existem comandos que dependem da manutenção de sessão (diálogo entre OBDH e PDC), e aqueles independentes de sessão.

5 PROJETO DOS COMPONENTES

Este capítulo descreve os componentes do projeto arquitetural do SWPDC, com o objetivo de orientar a construção do código-fonte para a plataforma de hardware e linguagem escolhida.

A notação textual das interfaces segue a mesma adotada pela UML, semelhante à sintaxe da linguagem Pascal. Os indicadores **IN**, **INOUT** e **OUT** representam o sentido (direção) dos parâmetros classificando como entrada, entrada e saída, e saída, respectivamente.
A estrutura dos tópicos a seguir cria uma enumeração estruturada de todas as interfaces e classes modeladas, contendo uma descrição sucinta, seguida pela definição de seus parâmetros de entrada, parâmetros de saída e valor retorno, classificadas por componente.

5.1 COMUNICAÇÃO

Nesta seção estão descritas as interfaces e classes que compõem o componente de **Comunicação**.

Responsabilidades do componente Comunicação incluem:

- Ocultar a complexidade relacionada a comunicação com os atores externos ao sistema como OBDH e EPP.
- Gerenciar recursos de comunicação como *buffers* de E/S e UARTs.
- Implementar os protocolos de comunicação PDC-OBDH e PDC-EPP, permitindo enviar e receber mensagens para as suas respectivas entidades parceiras.

5.1.1 INTERFACES

IComunicadorOBDH: Conjunto de operações para comunicação com o OBDH, permitindo receber comandos e enviar respostas de acordo com o protocolo de comunicação PDC-OBDH.

- `PUBLIC enviarResposta(IN rsp: void*, IN nbytes: int16u): void`
  - **Descrição** - Permite o envio de dados em resposta a comandos do OBDH.
  - **rsp** - Ponteiro para a origem dos dados da resposta.
  - **nbytes** - Tamanho em bytes dos dados de resposta.
  - **retorno** - Nenhum.
- `PUBLIC inserirCKS(INOUT buffer: int8u*, INOUT size: int16u): void`
  - **Descrição** - Calcula e insere o checksum nos dois últimos bytes do buffer passado como parâmetro.
  - **buffer** - Ponteiro do buffer que precisa do checksum calculado.
  - **size** - Tamanho total do buffer (inclusive com checksum).
  - **retorno** - Nenhum.

- `PUBLIC obterComando(): MsgOBDH*`
  - **Descrição** - Recupera o último comando enviado pelo OBDH.
  - **retorno** - Último comando enviado pelo OBDH.

IDriverUART: Operações para trabalhar com as portas seriais. Oferece métodos para configurar, receber e enviar bytes em portas seriais (UART).

- `PUBLIC desabilitarRxTO(): void`
  - **Descrição** - Desabilita o *timeout* na recepção de dados da porta serial.
  - **retorno** - Nenhum.
- `PUBLIC enviar(IN buf: int8*, IN bufsz: int16u):`
  - **Descrição** - Envia, de forma assíncrona baseada em interrupções da UART, um buffer de dados com tamanho definido.
  - **buf** - Ponteiro para o buffer de dados a ser enviado.
  - **bufsz** - Tamanho do buffer de dados a ser enviado.
  - **retorno** - Nenhum.
- `PUBLIC enviarByte(IN b: int8u): int8u`
  - **Descrição** - Envia um byte pela porta serial.
  - **b** - Byte a ser enviado pela porta serial.
  - **retorno** - Indicador de sucesso da operação: 0 - operação bem sucedida, EV_TO - ocorreu timeout ao tentar enviar o byte b pela porta serial.
- `PUBLIC habilitarRxTO(): void`
  - **Descrição** - Habilita o *timeout* na recepção de dados da porta serial.
  - **retorno** - Nenhum.
- `PUBLIC iniciarBuffer(IN buffer: int8u*, IN tamanho: int16u): void`
  - **Descrição** - Método que inicia o buffer que conterá os dados recebidos da porta serial.
  - **buffer** - Ponteiro para a área do buffer.
  - **tamanho** - Tamanho do buffer.
  - **retorno** - Nenhum.
- `PUBLIC obtIdxRD(): int16u`
  - **Descrição** - Obtém indexador de leitura da mensagem.
  - **retorno** - Indexador de leitura da mensagem.
- `PUBLIC receberByte(): int16u`
  - **Descrição** - Remove um byte do buffer de entrada da porta serial.

- **retorno** - Um valor no intervalo [0 .. 255] correspondendo ao byte lido do buffer da porta serial, ou EV_VAZIO se o buffer está vazio, ou EV_TO se o tempo de espera pela chagada do próximo está esgotado.

- `PUBLIC setup(IN bps: TaxaEnum, IN txTO: int16u, IN rxTO: int16u): void`
  - **Descrição** - Configura uma porta serial. A porta em questão é determinada pelo realizador da interface.
  - **bps** - A taxa de operação da porta especificado pela enumeração TaxaEnum.
  - **txTO** - Tempo máximo em milissegundos a esperar pelo envio de um byte usando a porta serial.
  - **rxTO** - Tempo máximo em milissegundos a esperar pela recepção de um conjunto bytes usando a porta serial. Deve ser considerado somente quando da invocação de habilitarRxTO() for executado e desconsiderado quando desabilitarRxTO() for executado.
  - **retorno** - Nenhum.

- `PUBLIC EV_TO: int16u = 1001`

  Constante indicadora de timeout na recepção do próximo byte.

- `PUBLIC EV_VAZIO: int16u = 1000`

  Constante indicadora de buffer vazio.

IMsgOBDH: Interface da estrutura de dados de uma mensagem de comandos do protocolo PDC-OBDH.

- `PUBLIC obtCSC(): int8u`
  - **Descrição** - Método que obtém CSC da mensagem.
  - **retorno** - CSC da mensagem.
- `PUBLIC obtChecksum(): int16u`
  - **Descrição** - Método que obtém checksum da mensagem.
  - **retorno** - Checksum da mensagem.
- `PUBLIC obtComprimento(): int16u`
  - **Descrição** - Método que obtém comprimento da mensagem.
  - **retorno** - Comprimento da mensagem.
- `PUBLIC obtDado(IN idx: int16u): int8u*`
  - **Descrição** - Método que obtém dado(s) da mensagem.
  - **idx** - Indexador do dado desejado.
  - **retorno** - Ponteiro para o dado da mensagem requerido.
- `PUBLIC obtEnderFinal(): int16u`

- **Descrição** - Método que obtém endereço final a partir do campo de dados de mensagens que o requer.
- **retorno** - Endereço final a partir do campo de dados da mensagem.

- `PUBLIC obtEnderInicial(): int16u`
  - **Descrição** - Método que obtém endereço inicial a partir do campo de dados de mensagens que o requer.
  - **retorno** - Endereço inicial a partir do campo de dados da mensagem.
- `PUBLIC obtInicio(): int8u`
  - **Descrição** - Método que obtém delimitador de início da mensagem.
  - **retorno** - Delimitador de início da mensagem. (0xEB)
- `PUBLIC obtOrigem(): int8u`
  - **Descrição** - Método que obtém identificador de origem da mensagem.
  - **retorno** - Identificador de origem da mensagem.
- `PUBLIC obtTempo(): int32u`
  - **Descrição** - Método que obtém tempo (ou NU) da mensagem.
  - **retorno** - Tempo (ou NU) da mensagem.
- `PUBLIC obtTipo(): int8u`
  - **Descrição** - Método que obtém tipo da mensagem.
  - **retorno** - Tipo da mensagem.
- `PUBLIC vrfChecksum(): int16u`
  - **Descrição** - Calcula a verificação de checksum.
  - **retorno** - A soma de todos os bytes da mensagem, exceto o próprio checksum.

5.1.2 CLASSES

ComunicadorEPP: Classe responsável pela coordenação da comunicação com os conjuntos EPP-HX. Delega o controle de envio periódico de comandos aos conjuntos EPP-HX à classe **DespachadorCmdEppHXI**. Assim, **ComunicadorEPP** fica responsável pela recepção das respostas e pela montagem dos pacotes completos a serem inseridos nos buffers apropriados pela classe **dados::GerenciadorDados**.

- `PRIVATE enviarCmdEppHXI(): void`
  - **Descrição** - Envia o comando ao conjunto EPP. O comando deve ser antes forjado no membro privado m_cmdEpp pelo método gerarCmdEppHXI().
  - **retorno** - Nenhum.
- `PUBLIC eppCmd(): void`
  - **Descrição** - Cordena o envio de solicitações de dados enviados aos EPPs.
  - **retorno** - Nenhum.

- `PUBLIC executar(): void`
  - **Descrição** - Ponto de entrada da tarefa.
  - **retorno** - Nenhum.
- `PRIVATE gerarCmdEppHXI(IN tipo: EventPackEnum, IN IdEHXi: IdEppHXEnum): void`
  - **Descrição** - Produz uma mensagem de comando EPP dado um tipo de pacote EPP e uma identificação do conjunto EPP-HX.
  - **tipo** - O tipo de dado do pacote EPP a ser requisitado.
  - **IdEHXi** - O conjunto EPP que será requisitado a responder à solicitação de dados.
  - **retorno** - Nenhum.

Atributos da classe **ComunicadorEPP**:

- `PRIVATE ctrTstHx1: int16u`

  Contador de Event Packets de Teste HX1.
- `PRIVATE ctrTstHx2: int16u`

  Contador de Event Packets de Teste HX1.
- `PRIVATE evPack: int8u`

  Representa a quantidade de bytes lidos de *EventPackets*.
- `PRIVATE m_cmdEpp: int8u`

  Representação da estrutura de dados de uma mensagem de comandos do protocolo PDC-EPP.
- `PRIVATE m_listaObs: IObservadorEventos`

  Lista de observadores de interessados na notificação de eventos da comunicação com EPP.
- `PRIVATE m_pacoteEHX: int8u`
- `PRIVATE m_tabTransicoes: TransicaoFsm`

  Tabela de transições de estado da máquina de estados finitos que reconhece respostas dos conjuntos EPP-HX.

ComunicadorOBDH: Implementa a interface *IComunicadorOBDH*. Mantém a máquina de estados finitos reconhecedora das mensagens de comando definidas pelo protocolo PDC-OBDH. Obtém os dados que compõem as mensagens por meio da classe **DriverOBDH**. Esses dados (bytes) alimentam a máquina de estados.

- `PRIVATE armazenaCKSLsb(IN ctx: EventoFsm): void`
  - **Descrição** - Armazena Lsb do checksum no buffer de mensagem recebida.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE armazenaCKSMsb(IN ctx: EventoFsm): void`

- **Descrição** - Armazena Msb do checksum no buffer de mensagem recebida.
- **ctx** - O contexto da ação.
- **retorno** - Nenhum.

- `PRIVATE armazenaCSC(IN ctx: EventoFsm): void`
  - **Descrição** - Armazena CSC no buffer de mensagem recebida.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE armazenaDado(IN ctx: EventoFsm): void`
  - **Descrição** - Armazena Dado no buffer de mensagem recebida.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE armazenaIO(IN ctx: EventoFsm): void`
  - **Descrição** - Armazena DI e IO no buffer de mensagem recebida.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE armazenaNU(IN ctx: EventoFsm): void`
  - **Descrição** - Armazena NU no buffer de mensagem recebida.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE armazenaTamLsb(IN ctx: EventoFsm): void`
  - **Descrição** - Armazena Lsb do Comprimento no buffer de mensagem recebida.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE armazenaTamMsb(IN ctx: EventoFsm): void`
  - **Descrição** - Armazena Msb do Comprimento no buffer de mensagem recebida.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE armazenaTipo(IN ctx: EventoFsm): void`
  - **Descrição** - Armazena Tipo no buffer de mensagem recebida.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.

- `PRIVATE cksOK(IN ctx: EventoFsm): int8u`
  - **Descrição** - Calcula e compara a soma de verificação de consistência da mensagem de comando.
  - **ctx** - O contexto da ação.
  - **retorno** - 1 - soma de verificação consistente; 0 - soma de verificação inconsistente.
- `PRIVATE cmdReqRspIme(): int8u`
  - **Descrição** - Verifica se o comando requer resposta imediata (COMANDO_RECEBIDO).
  - **retorno** - 1 - o comando requer resposta imediata; 0, caso contrário.
- `PRIVATE cndTrue(IN ctx: EventoFsm): int8u`
  - **Descrição** - Retorna sempre verdadeiro. Utilizado quando não se precisa verificar nenhuma condição.
  - **ctx** - O contexto da ação.
  - **retorno** - Retorna sempre TRUE.
- `PRIVATE desabilitarRxTO(IN ctx: EventoFsm): void`
  - **Descrição** - Chama o método desabilitarRxTO em DriverOBDH (desabilita *timeout*)
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PUBLIC executar(): void`
  - **Descrição** - Ponto de entrada da classe ativa **ComunicadorOBDH**.
  - **retorno** - Nenhum.
- `PRIVATE habilitarRxTO(IN ctx: EventoFsm): void`
  - **Descrição** - Chama o método habilitarRxTO em DriverOBDH (habilita *timeout*).
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE idNUOK(IN ctx: EventoFsm): int8u`
  - **Descrição** - Usado pela máquina de estados para identificar se o campo NU da mensagem de comando é válido (igual a zero).
  - **ctx** - O contexto da ação.
  - **retorno** - Indicador de sucesso de leitura dos quatro bytes do campo NU da mensagem de comando em análise: 1 - sucesso; 0 - insucesso (ainda não foram lidos os quatro bytes).

- `PRIVATE idOrigemOK(IN ctx: EventoFsm): int8u`
  - **Descrição** - Usado pela máquina de estados para identificar se o campo IO da mensagem de comando é válido.
  - **ctx** - O contexto da ação.
  - **retorno** - Indicador de validade do campo IO da mensagem de comando em análise: 1 - campo válido; 0 - campo inválido.
- `PRIVATE idTipoDdo(IN ctx: EventoFsm): IdTipoDdoEnum`
  - **Descrição** - Usado para categorizar o campo que indica o tamanho do campo de dados da mensagem de comando.
  - **ctx** - O contexto da ação.
  - **retorno** - Retorna informação sobre o tipo de dado: de tamanho zero, fixo ou variável.
- `PRIVATE ntfExeCmd(IN ctx: EventoFsm): void`
  - **Descrição** - Notifica aos observadores de evento que um comando acaba de ser reconhecido e está pronto para ser processado.
  - **ctx** - O contexto da ação - neste caso, representará a constante CTX_CMD_PRONTO em **pdcsys**.
  - **retorno** - Nenhum.
- `PRIVATE reiniciar(IN ctx: EventoFsm): void`
  - **Descrição** - Reinicia a máquina de estados.
  - **ctx** - O contexto da ação.
  - **retorno** - Nenhum.
- `PRIVATE tamDdoZer(IN ctx: EventoFsm): int8u`
  - **Descrição** - No caso de dados com tamanho fixo ou variável, obtém estes dados até o tamanho ficar igual a zero, e verifica viabilidade de sub-tipo em caso de tipo que o exige.
  - **ctx** - O contexto da ação.
  - **retorno** - 1 - se já foram lidos todos os bytes de dados; 0 - se ainda não foram lidos todos os dados.
- `PRIVATE tamFixOK(IN ctx: EventoFsm): int8u`
  - **Descrição** - Verifica se o campo de dados tem um tamanho fixo de acordo com o tipo da mensagem de comando.
  - **ctx** - O contexto da ação.
  - **retorno** - 1 - tamanho fixo válido; 0 - tamanho fixo inválido.
- `PRIVATE tamVarOK(IN ctx: EventoFsm): int8u`

- **Descrição** - Verifica se o campo de dados tem um tamanho variável de acordo com o tipo da mensagem de comando.
- **ctx** - O contexto da ação.
- **retorno** - 1 - tamanho variável válido; 0 - tamanho variável inválido.

- `PRIVATE tamZerOK(IN ctx: EventoFsm): int8u`
  - **Descrição** - Verifica se o campo de dados tem tamanho zero.
  - **ctx** - O contexto da ação.
  - **retorno** - 1 - tamanho zero válido; 0 - tamanho inválido (não zero).
- `PRIVATE tipoOK(IN ctx: EventoFsm): int8u`
  - **Descrição** - Usado pela máquina de estados para identificar se o campo TIPO da mensagem de comando é válido.
  - **ctx** - O contexto da ação.
  - **retorno** - Indicador de validade do campo TIPO da mensagem de comando em análise: 1 - campo válido; 0 - campo inválido.
- `PRIVATE tipoTamFix(IN ctx: EventoFsm): int8u`
  - **Descrição** - Usado para verificar se o tipo do comando é compatível com dados de tamanho fixo.
  - **ctx** - O contexto da ação.
  - **retorno** - 1 - se tipo é compatível com mensagem com dados de tamanho fixo; 0 - se tipo não o é compatível.
- `PRIVATE tipoTamVar(IN ctx: EventoFsm): int8u`
  - **Descrição** - Usado para verificar se o tipo do comando é compatível com dados de tamanho variável.
  - **ctx** - O contexto da ação.
  - **retorno** - 1 - se tipo é compatível com mensagem com dados de tamanho variável; 0 - se tipo não o é compatível.
- `PRIVATE tipoTamZer(IN ctx: EventoFsm): int8u`
  - **Descrição** - Usado para verificar se o tipo do comando é compatível com dados de tamanho zero.
  - **ctx** - O contexto da ação.
  - **retorno** - 1 - se tipo é compatível com mensagem sem dados; 0 - se tipo não o é compatível.

Atributos da classe **ComunicadorOBDH**:

- `PRIVATE m_fsm: FSM`

Instância da máquina de estados que implementará o reconhecimento das mensagens de comando do protocolo PDC-OBDH.

- `PRIVATE m_listaObs: IObservadorEventos`

  Observadores de eventos interessados em ser notificados quando da ocorrência de eventos relacionados à comunicação PDC-OBDH.

- `PRIVATE m_msgRX: MsgOBDH`

  Representa a última mensagem de comando do OBDH. Esse dado é volátil, no sentido de que seus campos podem estar sendo manipulados pela máquina de estados do protocolo PDC-OBDH. Portanto, seu uso pelas classes de serviço somente é liberado por meio de notificações de evento.

- `PRIVATE m_rspCmdRbd: int8u`

  Vetor de bytes de tamanho fixo que representa a mensagem de resposta COMANDO_RECEBIDO do protocolo PDC-OBDH.

- `PRIVATE m_tabTransicoes: TransicaoFsm`

  Tabela de transições de estado, iniciada por **ComunicadorOBDH** e usada pela **FSM**.

- `PRIVATE m_tamDdo: int16u`

  Valor temporarário para reconhecer e obter campos de dados de comprimento variável de certas mensagens de comando do OBDH.

DriverEPP: Implementa a interface *IDriverUART* para tratar a recepção e o envio de dados por meio das porta UART0 (PDC_EPP), usando o TIMER2.

- `PRIVATE uartISR(): void`
  - **Descrição** - Rotina de tratamento de interrupções associadas à UART0, por meio da qual os buffers do driver são processados.
  - **retorno** - Nenhum.

Atributos da classe **DriverEPP**:

- `PRIVATE m_buffer: BufferSimples`

  Gerenciador do buffer de entrada do protocolo PDC-EPP.

- `PRIVATE m_memBuff: int8u`

  Memória do buffer de entrada.

- `PRIVATE m_rxTO: int16u`

  Timeout de recepção de mensagem da porta serial.

- `PRIVATE m_rxTO_hab: int8u`

  Indica se o timeout de leitura deve ser considerado. Valores: 1- habilitado, 0 - desabilitado.

- `PRIVATE m_txTO: int16u`

  Timeout de escrita na porta serial.

DriverOBDH: Implementa a interface *IDriverUART* para tratar a recepção e o envio de dados por meio das porta UART1 (PDC_OBDH), usando o TIMER1.

- `PRIVATE uartISR(): void`
  - **Descrição** - Rotina de tratamento de interrupções associadas à UART1, por meio da qual os buffers do driver são processados.
  - **retorno** - Nenhum.

Atributos da classe **DriverOBDH**:

- `PRIVATE m_memBuf: int8u*`

  Memória do buffer de entrada.

- `PRIVATE m_rd: int16u`

  Indexador de leitura do buffer do Driver (m_memBuf).

- `PRIVATE m_rxTO: int16u`

  Timeout de recepção de mensagem da porta serial.

- `PRIVATE m_rxTO_hab: int8u`

  Indica se o timeout de leitura deve ser considerado. Valores: 1- habilitado, 0 - desabilitado.

- `PRIVATE m_tamBuf: int16u`
- `PRIVATE m_txTO: int16u`

  Timeout de escrita na porta serial.

- `PRIVATE m_wr: int16u`

  Indexador de escrita do buffer do Driver (m_memBuf)

MsgOBDH: Representação da estrutura de dados de uma mensagem de comandos do protocolo PDC-OBDH.

Atributos da classe **MsgOBDH**:

- `PUBLIC m_iniMsg: int16u`

  Marcador do início da mensagem dentro do buffer circular **m_msg**.

- `PUBLIC m_msg: int8u`

  Buffer circular que contém a mensagem.

PacoteEPP: Estrutura de dados para representar um pacote EPP.

Atributos da classe **PacoteEPP**:

- `PUBLIC m_eventos: int8u`

  Informações da carga útil do pacote EPP.

- `PUBLIC m_idEHX: int8u`

  Indicador da origem do pacote, ou seja, qual conjunto EPP-HX.

- `PUBLIC m_timestamp: int32u`

Marca temporal do pacote EPP.

EstadosFsmOBDHEnum: Enumeração dos estados da máquina de estados do protocolo PDC-OBDH.

- `{efoCKS, efoCSC, efoDI, efoDdoTamFix, efoDdoTamVar, efoDdoTamZer, efoIO, efoNU, efoObtDdo, efoTipo}`

IdEppHXEnum: Enumeração das identificações dos conjuntos EPP-HX.

- `{efoCKS, efoCSC, efoDI, efoDdoTamFix, efoDdoTamVar, efoDdoTamZer, efoIO, efoNU, efoObtDdo, efoTipo ieHX1 = 0x04, ieHX2 = 0x08}`

IdTipoDdoEnum: Enumeração dos tipos do campo de dado de uma mensagem no protocolo PDC-OBDH quanto ao seu tamanho.

- `{efoCKS, efoCSC, efoDI, efoDdoTamFix, efoDdoTamVar, efoDdoTamZer, efoIO, efoNU, efoObtDdo, efoTipo ieHX1 = 0x04, ieHX2 = 0x08idDdoTamFix, idDdoTamVar, idDdoTamZer}`

5.2 DADOS

O componente de dados possui as interfaces e classes necessárias para ocultar a complexidade do gerenciamento de dados como buffers e páginas de dados.

Responsabilidades principais:

- Gerenciar filas (buffers circulares).
- Gerenciar buffers dos diversos tipos de dados da aplicação.
- Oferecer interfaces para inclusão e remoção de dados nos buffers.

5.2.1 INTERFACES

IBuffer: Interface de operações comuns sobre buffers.

- `PUBLIC estaVazio(): int8u`
  - **Descrição** - Verifica se o buffer está vazio.
  - **retorno** - 1, indica que o buffer está vazio, 0, caso contrário.
- `PUBLIC inserir(IN dado: int8u*, IN tamanho: int16u): int16u`
  - **Descrição** - Insere um dado no buffer.
  - **dado** - Ponteiro para endereço do primeiro byte do dado a ser inserido no buffer.
  - **tamanho** - Tamanho do dado a ser inserido no buffer.
  - **retorno** - A quantidade de dados efetivamente gravada no buffer.
- `PUBLIC remover(IN dado: int8u*, IN tamanho: int16u): int16u`
  - **Descrição** - Remove dados do buffer.
  - **dado** - Ponteiro para área de destino dos dados a serem removidos do buffer.
  - **tamanho** - Quantidade de bytes a serem removidos.
  - **retorno** - A quantidade de bytes efetivamente removida do buffer.

- `PUBLIC tamanho(): int32u`
  - **Descrição** - Retorna o tamanho do buffer.
  - **retorno** - O tamanho do buffer em bytes.

IBufferSimples: Interface para buffers que manipulam memória de área **comum a todas páginas** (memória *estática*, não paginada).

- `PUBLIC alocar(IN area: void*, IN tamanho: int16u): void`
  - **Descrição** - Registra a alocação de uma área de memória para ser manipulada pelo buffer.
  - **area** - Ponteiro para o início da memória alocada.
  - **tamanho** - Tamanho da área alocada a ser registrada.
  - **retorno** - Nenhum.

IBufferVirtual: Interface de operações sobre buffers paginados, que residem no espaço de memória paginada.

- `PUBLIC iniciar(IN memoria: MemoriaVirtual*, IN tamBloco: int16u, IN apagavel: int8u): void`
  - **Descrição** - Inicia o buffer virtual com um conjunto de páginas identificadas pela região de memória indicada por **MemoriaVirtual***
  - **memoria** - Região (segmento de página, página, setores) de memória virtual que será alocada para o buffer.
  - **tamBloco** - Tamanho do bloco a ser gravado.
  - **apagavel** - 1 (TRUE), se o buffer tem autonomia para apagar a si mesmo a cada overlap; ou 0 (FALSE) caso contrário.
  - **retorno** - Nenhum.

IFormatador: Interface de formatação de dados para transmissão. Usada para definir a montagem de mensagens de resposta no formato definido pelo protocolo PDC-OBDH. A formatação corre em duas fases. Primeiro seleciona-se o método formatador (um para cada tipo de resposta no protocolo PDC-OBDH), depois obtém-se o buffer com os dados formatados, prontos para transmissão.

- `PUBLIC fmtRspCmdRec(): void`
  - **Descrição** - Monta uma mensagem de resposta do tipo 81h - Comando recebido;.
  - **retorno** - Nenhum.
- `PUBLIC fmtRspDdoRgo(IN rgo: int32u): void`
  - **Descrição** - Monta uma mensagem de resposta do tipo 82h - Dados de relógio.
  - **rgo** - Tempo atual.
  - **retorno** - Nenhum.

- `PUBLIC fmtRspNoData(): void`
  - **Descrição** - Monta uma mensagem de resposta do tipo 84h - Nenhum dado.
  - **retorno** - Nenhum.
- `PUBLIC fmtRspStaCga(IN subtipo: int8u, IN valor: int8u): void`
  - **Descrição** - Monta uma mensagem de resposta do tipo 8Ah - Status da carga.
  - **subtipo** - Subtipo do status.
  - **valor** - Valor do status.
  - **retorno** - Nenhum.
- `PUBLIC fmtRspTxDdo(IN subTipo: int8u, IN csr: int16s): void`
  - **Descrição** - Monta uma resposta para o pedido de transmissão de dados.
  - **subTipo** - Subtipo dos dados a serem transmitidos.
  - **csr** - CSR - quantas respostas ainda estão disponíveis.
  - **retorno** - Nenhum.
- `PUBLIC fmtRspVrfModOpe(IN modo: ModoOperacaoEnum): void`
  - **Descrição** - Monta uma mensagem de resposta do tipo 83h - Informação de modo de operação.
  - **modo** - Modo de operação corrente.
  - **retorno** - Nenhum.
- `PUBLIC obtBufRsp(): int8u*`
  - **Descrição** - Obtém o do buffer com a última mensagem de resposta formatada.
  - **retorno** - Buffer de resposta formatado.
- `PUBLIC obtTamBufRsp(): int16u`
  - **Descrição** - Obtém o tamanho do buffer da última mensagem de resposta formatada.
  - **retorno** - Tamanho do último buffer formatado.

IGerenciadorDados: Operações de gerenciamento de buffers do sistema.

- `PUBLIC adicionar(IN destino: TipoBufferEnum, IN dado: int8u*, IN tamanho: int16u): int16u`
  - **Descrição** - Adiciona dados a um buffer específico.
  - **destino** - Indicador do buffer de destino, o qual receberá os dados.
  - **dado** - Ponteiro para o início da área a ser copiada (inserida) no buffer.
  - **tamanho** - A quantidade de bytes a ser inserida no buffer.

- **retorno** - A quantidade de bytes efetivamente inserida no buffer.

- `PUBLIC altBufDmp(IN endInicial: int16u, IN endFinal: int16u): void`
  - **Descrição** - Altera ponteiros do buffer de descarga de memória.
  - **endInicial** - Endereço Inicial.
  - **endFinal** - Endereço Final.
  - **retorno** - Nenhum.
- `PUBLIC altEnderecoInicialDmp(IN endInicial: int16u): void`
  - **Descrição** - Altera endereço inicial do buffer de descarga.
  - **endInicial** - Endereço Inicial.
  - **retorno** - Nenhum.
- `PUBLIC altSelecaoMemDmp(IN mem: int8u): void`
  - **Descrição** - Altera seleção de memória que se pretende fazer dump.
  - **mem** - Indicador da memória a ser selecionada.
  - **retorno** - Nenhum.
- `PUBLIC iniciar(): void`
  - **Descrição** - Efetua os procedimentos de iniciação do gerenciador de dados.
  - **retorno** - Nenhum.
- `PUBLIC iniciarCga(IN endereco: int16u): void`
  - **Descrição** - Efetua os procedimentos de iniciação do buffer de carga.
  - **endereco** - Endereço inicial para carga.
  - **retorno** - Nenhum.
- `PUBLIC iniciarDgeHx2(): void`
  - **Descrição** - Efetua os procedimentos de iniciação do buffer de diagnóstico da câmera HXI 2.
  - **retorno** - Nenhum.
- `PUBLIC iniciarDmp(): void`
  - **Descrição** - Efetua os procedimentos de iniciação do buffer de descarga de memória.
  - **retorno** - Nenhum.
- `PUBLIC iniciarTst(): void`
  - **Descrição** - Efetua os procedimentos de iniciação do buffer de teste.
  - **retorno** - Nenhum.
- `PUBLIC obtEnderecoInicialDmp(): int16u`

- **Descrição** - Obtém endereço inicial do buffer de descarga.
- **retorno** - O valor do endereço inicial do buffer de descarga.

- `PUBLIC obtSelecaoMemDmp(): int8u`
  - **Descrição** - Obtém seleção de memória que se pretende fazer dump.
  - **retorno** - O valor do indicador da memória selecionada.
- `PUBLIC obterCSR(IN subtipo: int8u): int16s`
  - **Descrição** - Obtém quantidade de pacotes ainda disponíveis.
  - **subtipo** - Subtipo dos dados de resposta.
  - **retorno** - O valor do CSR (Controle de Seqüência de Resposta) corrente.
- `PUBLIC obterUltimoIdHx(): int8u`
  - **Descrição** - Retorna último indicador de pulling dos conjuntos EPP Hi - HXi (idHX).
  - **retorno** - Valor do indicador idHXi.
- `PUBLIC relatarEvento(IN tipo: TipoRelatoEventoEnum, IN info: int8u[5]): void`
  - **Descrição** - Insere no buffer de relato de eventos um dado evento, juntamente com sua informação associada.
  - **tipo** - Indicador do tipo do relato de evento.
  - **info** - Informação associada ao evento. Deve ser obrigatoriamente um vetor de 5 bytes.
  - **retorno** - Nenhum.
- `PUBLIC remover(IN local: TipoBufferEnum, IN dado: int8u*, IN tamanho: int16u): int16u`
  - **Descrição** - Remove dados de um buffer específico.
  - **local** - Indicador do buffer de origem dos dados a serem removidos.
  - **dado** - Ponteiro para a área de memória de destino, o qual serão copiados os bytes removidos do buffer.
  - **tamanho** - Quantidade de bytes a serem removidos do buffer.
  - **retorno** - A quantidade de bytes efetivamente removida do buffer.

5.2.2 CLASSES

BufferSimples: Classe que implementa a interface *IBufferSimples*.

Atributos da classe **BufferSimples**:

- `PRIVATE m_area: void*`

  Ponteiro para a área de dados alocada para este buffer.

- `PRIVATE m_ixrd: int16u`

  Indexador de leitura do buffer.

- `PRIVATE m_ixwr: int16u`

  Indexador de escrita do buffer.

- `PRIVATE m_tamanho: int16u`

  Tamanho da em bytes da área de dados alocada.

BufferVirtual: Buffer de memória. Usado para manipular dados (bytes) em memória. Um Buffer não sabe nada a respeito do conteúdo da memória que ele manipula. A estratégia de manipulação dados é do tipo buffer circular (*ring-buffer*).

- `PRIVATE gravarFlash(IN ender: EnderecoVirtual, IN dado: int8u): int8u`
  - **Descrição** - Dado um endereço de memória externa (XDATA) da página corrente e um byte, grava o byte na memória *flash* (endereço dado). Atenção: Antes de chamar este método, a página de dados já deve ter sido selecionada.
  - **ender** - Endereço virtual da memória flash a ser gravado.
  - **dado** - Byte a ser gravado (programado) na memória flash.
  - **retorno** - Retorna 0 (zero) se houve erro na operação de escrita do byte na memória flash; ou 1 (um) se a operação foi bem sucedida.

Atributos da classe **BufferVirtual**:

- `PRIVATE apagavel: int8u`

  1, se o buffer tem autonomia para apagar a si mesmo a cada overlap.

- `PRIVATE flgPagErase: int8u`

  1, se ocorreu erase na página corrente.

- `PRIVATE m_buf: MemoriaVirtual*`

  Referência para uma região de memória virtual a ser manipulada pelo buffer.

- `PRIVATE m_ixrd: EnderecoVirtual`

  Endereço virtual de referência para o indexador de escrita do buffer a partir do qual dados podem ser escritos.

- `PRIVATE m_ixwr: EnderecoVirtual`

  Endereço virtual de referência para o indexador de escrita do buffer a partir do qual dados podem ser escritos.

- `PRIVATE tamBloco: int16u`

  Tamanho do Bloco a ser gravado.

Endereco: Representação de um endereço virtual em página e deslocamento dentro da página.

Atributos da classe **Endereco**:

- `PUBLIC paginaLSB: int8u`

Byte menos significativo do endereço da página. De acordo com o esquema de paginação utilizado nessa versão do SWPDC, esse byte assumirá valores de 0 a 7, indicando qual página deverá ser referenciada.

- `PUBLIC paginaMSB: int8u`

  Byte mais significativo do endereço da página. De acordo com o esquema de paginação utilizado nessa versão do SWPDC, esse byte será sempre zero.

- `PUBLIC valor: int16u`

  Endereço dentro da página selecionada. De acordo com o esquema de paginação utilizado, o endereço poderá variar de 8000h à FFFFh.

EnderecoFisico: Um endereço virtual é um inteiro de 32bits que pode ser interpretado de forma linear ou divido em dois endereços: página e deslocamento (offset) dentro da página.

Atributos da classe **EnderecoFisico**:

- `PUBLIC end32bit: int32u`

  Interpretação de um endereço virtual como um inteiro de 32 bits.

- `PUBLIC endereco: Endereco`

  Interpretação de um endereço virtual separado em página (MSB e LSB) e endereço (deslocamento) dentro da página.

Formatador: Implementa a interface *IFormatador*. Provê funcionalidades de montagem de mensagens de resposta no formato definido pelo protocolo PDC-OBDH.

Atributos da classe **Formatador**:

- `PRIVATE m_fmtbuf: int8u`

  Buffer alocado para montagem das respostas.

- `PRIVATE m_tambuf: int16u`

  Tamanho efetivo em bytes do buffer alocado para montagem das respostas. Se diferente de zero, indica a presença da última mensagem formatada.

GerenciadorDados: Coordena a manipulação de dados dos buffers: científico, diagnóstico, housekeeping, carga de dados, descarga de dados e relato de eventos.

Atributos da classe **GerenciadorDados**:

- `PRIVATE m_bufCco: IBufferVirtual`

  Buffer virtual para dados científicos.

- `PRIVATE m_bufCga: IBufferVirtual`

  Buffer virtual para carga de dados/programas.

- `PRIVATE m_bufDge: IBufferVirtual`

  Buffer virtual para dados de diagnóstico.

- `PRIVATE m_bufDmp: IBufferVirtual`

  Buffer virtual para descarga de dados - *dump*.

- `PRIVATE m_bufEvt: IBufferSimples`

  Buffer simples (memória não paginada) para relato de eventos.

- `PRIVATE m_bufHke: IBufferVirtual`

  Buffer virtual para dados de *housekeeping*.

- `PRIVATE m_bufTstHX1: IBufferVirtual`

  Buffer virtual para dados de teste da EPP HX1.

- `PRIVATE m_bufTstHX2: IBufferVirtual`

  Buffer virtual para dados de teste da EPP HX2.

- `PRIVATE m_memCco: MemoriaVirtual`

  Região de memória virtual para dados científicos.

- `PRIVATE m_memCga: MemoriaVirtual`

  Região de memória virtual para carga de dados/programa.

- `PRIVATE m_memDge: MemoriaVirtual`

  Região de memória virtual para dados de diagnóstico.

- `PRIVATE m_memDmp: MemoriaVirtual`

  Região de memória virtual para descarga de dados - *dump*.

- `PRIVATE m_memHke: MemoriaVirtual`

  Região de memória virtual para dados de *housekeeping*.

- `PRIVATE m_memRelEvt: RelatoEvento`

  Área de memória para os relatos de eventos.

- `PRIVATE m_memTstHX1: MemoriaVirtual`

  Região de memória virtual para dados de teste da EPP HX1.

- `PRIVATE m_memTstHX2: MemoriaVirtual`

  Região de memória virtual para dados de teste da EPP HX2.

Housekeeper: Tarefa responsável pela geração de dados de *housekeeping*.

- `PUBLIC executar(): void`
  - **Descrição** - Ponto de entrada da tarefa.
  - **retorno** - Nenhum.
- `PRIVATE prepararPacoteHK(): void`
  - **Descrição** - Executa a preparação do pacote de *housekeeping*, acessando as origens de dados para um *housekeeping* completo, gravando os dados no pacote de *housekeeping* interno (m_pacHke).
  - **retorno** - Nenhum.

Atributos da classe **Housekeeper**:

- `PRIVATE m_pacHke: PacoteHK`

Memória de trabalho para composição do pacote de housekeeping.

MemoriaVirtual: Representação de um fragmento (ou região) de memória virtual paginada - memória flash. Contém os endereços inicial e final da região. Uma região de memória virtual é manipulada por um BufferVirtual.

Atributos da classe **MemoriaVirtual**:

- `PUBLIC endFinal: EnderecoVirtual`

  O endereço final da região de memória virtual.

- `PUBLIC endInicial: EnderecoVirtual`

  O endereço inicial da região de memória virtual.

PacoteHK: Representação de um pacote de dados de *Housekeeping*.

Atributos da classe **PacoteHK**:

- `PUBLIC amoTta: int8u`

  Últimas 120 amostras de temperaturas, 60 amostras por cada ponto (1 e 2), intercalados (10 bits por amostra).

- `PUBLIC modOpe: int8u`

  Modo de operação corrente do SWPDC.

- `PUBLIC ptrCga: int16u`

  Ponteiro de carga de programas.

- `PUBLIC qteErrCG: int8u`

  Quantidade de erros de cão-de-guarda ocorridos desde o último zeramento.

- `PUBLIC qteErrDup: int8u`

  Quantidade de erros duplos ocorridos desde o último zeramento.

- `PUBLIC qteErrSim: int8u`

  Quantidade de erros simples ocorridos desde o último zeramento.

- `PUBLIC relEvt: int8u`

  Últimos relatos de eventos ocorridos.

- `PUBLIC staAlmEHX1: int8u`

  Status da alimentação do conjunto EPP-HX1.

- `PUBLIC staAlmEHX2: int8u`

  Status da alimentação do conjunto EPP-HX2.

- `PUBLIC tamanho: int16u`

  Tamanho dos dados de *housekeeping*.

- `PUBLIC tempoATual: int32u`

  Tempo em que é formado o pacote (que será estampado no pacote de *housekeeping*)

- `PUBLIC tpoAmoTta: int16u`

  Tempo de amostragem de temperaturas.

- `PUBLIC tpoHke: int16u`

  Tempo de *housekeeping* em segundos.

RelatoEvento: Representação de um relato de evento.

Atributos da classe **RelatoEvento**:

- `PUBLIC idTipo: int8u`

  Indicador do tipo do relato de evento.

- `PUBLIC info: char`

  Informação associada ao relato de evento.

- `PUBLIC timestamp: int32u`

  Tempo do sistema quando da geração do relato de evento.

Temperatura: Tarefa responsável pela medição das temperaturas nos pontos 1 e 2 (portas ADC_C1 e ADC_C2).

- `PUBLIC executar(): void`
  - **Descrição** - Ponto de entrada da tarefa.
  - **retorno** - Nenhum.
- `PRIVATE iniciarCanal(IN canal: int8u): void`
  - **Descrição** - Inicia canal ADC.
  - **canal** - Canal ADC usado para ler a temperatura (ponto para amostrar digitalmente a tensão aplicada no canal).
  - **retorno** - Nenhum.
- `PRIVATE lerTemperatura(IN canal: int8u): int16u`
  - **Descrição** - Lê a temperatura.
  - **canal** - Canal ADC usado para ler a temperatura (ponto para amostrar digitalmente a tensão aplicada no canal).
  - **retorno** - Valor digital correspondente a tensão lida do conversor AD no canal especificado.
- `PUBLIC obterAmostras(IN buf: int8u[150]): void`
  - **Descrição** - Obtém as últimas amostras de temperaturas, formatadas em 10 bits por amostra.
  - **buf** - Ponteiro para um vetor de tamanho fixo de 150 bytes, para comportas as últimas amostras 160 de temperaturas, em 10 bits por amostra.
  - **retorno** - Nenhum.

Atributos da classe **Temperatura**:

- `PRIVATE m_bufTta: BufferSimples`

Referência para o Gerenciador de Dados.

- `PRIVATE m_buffer: int8u`

  Últimas 60 amostras de cada ponto de leitura de temperatura.

- `PRIVATE m_ixbuf: int8u`

  Indexador da próxima posição livre do buffer (circular) de temperaras.

TipoBufferEnum: Categorias de buffers de dados que podem ser manipulados pelo sistema.

- `{tbCientifico, tbDiagnose, tbHousekeeping, tbRelatoEventos, tbTemperatura, tbTeste}`

TipoRelatoEventoEnum: Enumeração dos códigos dos tipos de relato de evento que podem ser gerados pelo sistema.

- `{tbCientifico, tbDiagnose, tbHousekeeping, tbRelatoEventos, tbTemperatura, tbTestetreBufDdoCco10, treBufDdoCco90, treBufDdoDga10, treBufDdoDga90, treBufDdoDge10, treBufDdoDge90, treBufDdoHke10, treBufDdoHke90, treComEP_CKS, treComEP_DI, treComEP_EvPack, treComEP_IdD, treComEP_IdO, treComEP_Timeout, treComOP_CKS, treComOP_CSC, treComOP_Com, treComOP_DI, treComOP_Dados, treComOP_IO, treComOP_Timeout, treComOP_Tipo, treEHXI2Off, treEHXI2On, treEHXIOff, treEHXIOn, treEndCgaDifPtrCga, treEndIniIguFinDga, treEndIniMaiFinDga, treErrCG, treErrDup, treErrParModProComOBDH, treErrSim, treErrSubAtuProComOBDH, treErrTipProComOBDH, treLimMemPri, treManVarCfg, treModOpePDC, treOpeCGPOST, trePtrIniCga, trePwrON, treRST, treStaAlmEHXIPOST, treStaAlmPDCPOST = 1, treTtaFoaLim, treTtaPDCPOST, treTxDdoCfoHxOff, treTxDdoHxIni, treVrfMemPrg, treVrfMemSRAM}`

5.3 ESTADO

Nesta seção estão descritas as interfaces e classes que compõem o componente de estado.

Responsabilidades do componente **estado** incluem:

- Ocultar e gerenciar os dados de configuração do PDC.
- Monitorar o circuito de cão-de-guarda.
- Confinar e recuperar erros.

5.3.1 INTERFACES

IAcaoFsm: Representa uma ação a ser executada na transição de um estado.

- `PUBLIC acao(IN contexto: int8u): void`
  - **Descrição** - Executa a ação associada ao disparo de uma transição.
  - **contexto** - Identificador de contexto da ação cuja semântica é dada pela própria ação.
  - **retorno** - Nenhum.

ICondicaoFsm: Representa uma condição para verificar se uma transição pode ser disparada.

- `PUBLIC condicao(IN contexto: int8u): int8u`

- **Descrição** - Executa a avaliação da condição de disparo de uma transição num dado contexto.
- **contexto** - Identificador de contexto da condição.
- **retorno** - 0 (zero), se a condição falhou; ou outro valor caso a condição seja verdadeira.

IConfiguracao: Interface de acesso às configurações do SWPDC.

- `PUBLIC altFlgIniPdc(IN flag: int8u): void`
  - **Descrição** - Altera o indicador de iniciação do PDC.
  - **flag** - O indicador de iniciação.
  - **retorno** - Nenhum.
- `PUBLIC altModOpePdc(IN modo: ModoOperacaoEnum): void`
  - **Descrição** - Altera o modo de operação corrente do PDC.
  - **modo** - Modo de operação.
  - **retorno** - Nenhum.
- `PUBLIC altPtrCgaDdo(IN ptr: int16u): void`
  - **Descrição** - Altera o ponteiro inicial de carga de dados.
  - **ptr** - Ponteiro inicial de carga.
  - **retorno** - Nenhum.
- `PUBLIC altTipEvtPacAdq(IN tipoEvent: EventPackEnum): void`
  - **Descrição** - Altera o tipo de dados a adquirir via EPP.
  - **tipoEvent** - O tipo desejado.
  - **retorno** - Nenhum.
- `PUBLIC altTpoAmoTta(IN tempo: int16u): void`
  - **Descrição** - Altera o tempo de amostragem de temperaturas.
  - **tempo** - Tempo de amostragem de temperatuas em segundos.
  - **retorno** - Nenhum.
- `PUBLIC altTpoExeHke(IN tempo: int16u): void`
  - **Descrição** - Altera o tempo de execução de housekeeping.
  - **tempo** - Tempo de execução em segundos.
  - **retorno** - Nenhum.
- `PUBLIC incErrCG(): void`
  - **Descrição** - Incrementa o contador de erros de cão-de-guarda em uma unidade.

- **retorno** - Nenhum.

- `PUBLIC incErrDup(): void`
  - **Descrição** - Incrementa o contador de erros duplos em uma unidade.
  - **retorno** - Nenhum.
- `PUBLIC incErrSim(): void`
  - **Descrição** - Incrementa o contador de erros simples em uma unidade.
  - **retorno** - Nenhum.
- `PUBLIC iniDdoCfgPdc(): void`
  - **Descrição** - Inicia os dados configuração com seus valores padrão.
  - **retorno** - Nenhum.
- `PUBLIC obtFlgAdqEvtPac():`
  - **Descrição** - Obtém o indicador de aquisição de dados em andamento.
  - **retorno** - Zero: não adquirindo. Outro valor: aquisição em progresso.
- `PUBLIC obtFlgIniPdc(): int8u`
  - **Descrição** - Obtém o indicador de iniciação do PDC.
  - **retorno** - Zero: iniciado via Power-On. Outro valor: iniciado via reset.
- `PUBLIC obtModOpePdc(): ModoOperacaoEnum`
  - **Descrição** - Obtém o modo de operação corrente.
  - **retorno** - O modo de operação corrente do SWPDC.
- `PUBLIC obtPtrCgaDdo(): int16u`
  - **Descrição** - Obtém o valor do ponteiro inicial de carga de dados.
  - **retorno** - Valor do ponteiro.
- `PUBLIC obtQteErrCG(): int8u`
  - **Descrição** - Obtém a quantidade de erros de cão-de-guarda.
  - **retorno** - Retorna a quantidade de erros de cão-de-guarda.
- `PUBLIC obtQteErrDup(): int8u`
  - **Descrição** - Obtém a quantidade de erros duplos.
  - **retorno** - Retorna a quantidade de erros duplos.
- `PUBLIC obtQteErrSim(): int8u`
  - **Descrição** - Obtém a quantidade de erros simples.
  - **retorno** - Retorna a quantidade de erros simples.
- `PUBLIC obtTipEvtPacAdq():`
  - **Descrição** - Obtém o tipo de dados da aquisição corrente.

- **retorno** - Tipo de dado a adquirir.

- `PUBLIC obtTpoAmoTta(): int16u`
  - **Descrição** - Obtém o tempo de amostragem de temperaturas atual.
  - **retorno** - Valor do tempo em segundos.
- `PUBLIC obtTpoExeHke(): int16u`
  - **Descrição** - Obtém o tempo de execução de *housekeeping*.
  - **retorno** - Tempo de execução em segundos.
- `PUBLIC vrfRgaCmdModOpe(IN tipCmd: int8u): bflag`
  - **Descrição** - Verifica a regra de operação para um dado tipo de comando em função do modo de operação corrente.
  - **tipCmd** - Tipo do comando a ser verificado com o modo de operação corrente.
  - **retorno** - TRUE, se permite executar comando no modo de operação corrente; ou FALSE caso contrário.
- `PUBLIC vrfRgaCmdModOpeSub(IN subTipCmd: int8u): bflag`
  - **Descrição** - Verifica a regra de operação para um dado subtipo do comando Transmitir Dados em função do modo de operação corrente.
  - **subTipCmd** - Subtipo do comando Transmitir Dados a ser verificado com o modo de operação corrente.
  - **retorno** - TRUE, se permite transmissão com subtipo do comando no modo de operação corrente; ou FALSE caso contrário.
- `PUBLIC zerErrCG(): void`
  - **Descrição** - Zera o contador de erros de cão-de-guarda.
  - **retorno** - Nenhum.
- `PUBLIC zerErrDup(): void`
  - **Descrição** - Zera o contador de erros duplos.
  - **retorno** - Nenhum.
- `PUBLIC zerErrSim(): void`
  - **Descrição** - Zera o contador de erros simples.
  - **retorno** - Nenhum.

IFSM: Operações para manipular máquinas de estados finitos.

- `PUBLIC iniciar(IN inicial: EstadoFsm, IN erro: EstadoFsm, IN trans: TransicaoFsm*, IN ntrans: int16u): void`
  - **Descrição** - Inicia o mecanismo de máquina de estados finitos, a partir do estado inicial da máquina, uma tabela de transição previamente configurada e, o número de elementos (linhas) da tabela de transição.

- **inicial** - O estado inicial da FSM.
- **erro** - O estado de erro.
- **trans** - A tabela de transições da FSM.
- **ntrans** - Quantidade de linhas da tabela de transições.
- **retorno** - Nenhum.

- `PUBLIC proximoEstado(IN evt: EventoFsm): EstadoFsm`
  - **Descrição** - Realiza a pesquisa pelo próximo estado na tabela de transição, em função do estado corrente e um identificador de evento.
  - **evt** - Identificador de evento.
  - **retorno** - O próximo estado, que ao retornar já será o estado atual, configurado automaticamente.
- `PUBLIC reiniciar(): void`
  - **Descrição** - Reinicia a máquina de estados finitos.
  - **retorno** - Nenhum.
- `PUBLIC sinalizarErro(): void`
  - **Descrição** - Coloca a FSM em estado erro, fazendo com que a pesquisa de transições de para próximo estado sempre retorne o estado de erro.
  - **retorno** - Nenhum.

5.3.2 CLASSES

Configuracao:

Implementação da interface de configuração do SWPDC.

Atributos da classe **Configuracao**:

- `PRIVATE m_flgAdqEvtPac: FlagAquisicaoEnum`

  Indica se há uma aquisição de *Event Packs* em andamento.
- `PRIVATE m_flgIniPdc: int8u`

  Flag de iniciação do PDC. Zero, indica iniciação via *Power-On*. Outro valor, iniciação via reset.
- `PRIVATE m_modOpePdc: ModoOperacaoEnum`

  Modo de operação corrente do PDC.
- `PRIVATE m_ptrCgaDdo: int16u`

  Ponteiro (endereço absoluto de memória) para carga de dados.
- `PRIVATE m_qteErrCG: int8u`

  Contador de erros de cão-de-guarda.
- `PRIVATE m_qteErrDup: int8u`

Contador de erros duplos.

- `PRIVATE m_qteErrSim: int8u`

  Contador de erros simples.

- `PRIVATE m_rgaOpe: RegraOperacao`

  Tabela de regra para mapear tipo de comandos por modo de operação.

- `PRIVATE m_staNovPrg: int8u`

  Indicador da carga de um novo programa; usado na eventual atualização do firmware (IAP - In *Application Programming*). Este atributo deve ser alocado no último byte endereçável da memória volátil (SRAM), no caso (**XDATA_START_ADDR + 0x1FFF**).

- `PRIVATE m_tipEvtPacAdq: TipoEventPackEnum`

  Tipo de dado (Event Pack) a ser adquirido pelo EPP.

- `PRIVATE m_tpoAmoTta: int16u`

  Tempo de amostragem de temperaturas em segundos.

- `PRIVATE m_tpoExeHke: int16u`

  Tempo de execução de *housekeeping* em segundos.

EstadoFsm: Um número natural (inteiro positivo ou zero).

EventoFsm: Um número natural (inteiro positivo ou zero).

FSM: Implementa a interface *IFSM* criando um mecanismo de máquina de estados finitos.

- `PRIVATE pesquisarTransicao(IN estado: EstadoFsm, IN evento: EventoFsm): TransicaoFsm*`
  - **Descrição** - Efetua uma busca na tabela de transições para identificar uma transição de estado em função da tripla (estado, evento, condição). Ao encontrar uma transição compatível, executa a ação relacionada - se houve - e devolve o próximo estado. No entanto, se uma condição de erro foi sinalizada, o estado **m_estadoErro** sempre será retornado.
  - **estado** - Um estado da máquina.
  - **evento** - Um evento.
  - **retorno** - Ponteiro para a transição de estado encontrada, ou NULL se não encontrou uma transição compatível.

Atributos da classe **FSM**:

- `PRIVATE m_estadoAtual: EstadoFsm`

  O estado corrente da máquina.

- `PRIVATE m_estadoErro: EstadoFsm`

  Um estado para confinamento de erro na máquina. Se a máquina entrar neste estado, a classe usuária da máquina deverá reiniciá-la.

- `PRIVATE m_estadoInicial: EstadoFsm`

O estado inicial da máquina.

- `PRIVATE m_numTransicoes: int16u`

  Quantidade de transições na tabela de transições.

- `PRIVATE m_staFSM: int8u`

  O estado da máquina de estados finitos: 1 (TRUE) significa que a máquina está não está em condição de erro; 0 (FALSE) significa que houve uma sinalização de condição de erro. Inicialmente seu valor é TRUE.

- `PRIVATE m_tabTransicoes: TransicaoFsm`

  Referência para a tabela de transições.

MonitorCG: Tarefa de monitoramento do circuito de cão-de-guarda.

- `PUBLIC executar(): void`
  - **Descrição** - Ponto de entrada da tarefa.
  - **retorno** - Nenhum.
- `PUBLIC pararTarefas(): void`
  - **Descrição** - Para todas as tarefas criadas.
  - **retorno** - Nenhum.
- `PUBLIC reset(): void`
  - **Descrição** - Reinicia o sistema.
  - **retorno** - Nenhum.

Atributos da classe **MonitorCG**:

- `PRIVATE m_obsCG: IObservadorEventos`

RegraOperacao: Estrutura da tabela de regras de operação. Relaciona tuplas (Comando, Modo de Operação) em valores booleanos [true, false], indicando a permissão de executar o comando no modo de operação especificado.

Atributos da classe **RegraOperacao**:

- `PUBLIC flgOpe: int8u`

  Indicador de permissão para executar a operação (comando). não-zero: permite, zero: não permite.

- `PUBLIC modOpe: ModoOperacaoEnum`

  Modo de operação.

- `PUBLIC tipCmd: int8u`

  Tipo do comando.

TransicaoFsm: Uma entrada de tabela de transição de estado.

Atributos da classe **TransicaoFsm**:

- `PUBLIC acao: IAcaoFsm`

A ação a ser executada antes da mudança de estado.

- `PUBLIC condicao: ICondicaoFsm`

  A condição a ser verificada.

- `PUBLIC estado: EstadoFsm`

  O estado corrente.

- `PUBLIC evento: EventoFsm`

  O evento ocorrido.

- `PUBLIC proximoEstado: EstadoFsm`

  O próximo estado (efetivação da mudança de estado).

EventPackEnum: Enumeração dos tipos de dados que podem ser adquiridos pelos EPPs.

- `{epCientifico = 1, epDiagnostico = 2, epTeste = 4}`

FlagAquisicaoEnum: Enumeração do estado da aquisição de dados.

- `{epCientifico = 1, epDiagnostico = 2, epTeste = 4faAdquirir = 1, faNaoAdquirir = 0}`

ModoOperacaoEnum: Enumeração dos modos de operação possíveis do PDC.

- `{epCientifico = 1, epDiagnostico = 2, epTeste = 4faAdquirir = 1, faNaoAdquirir = 0moDiagnostico, moIniciacao, moNominal, moSeguranca}`

5.4 SERVICOS

Esta parte descreve interfaces e classes que compõem o componente de serviços.

Responsabilidades principais:

- Coordenar a iniciação do PDC.
- Execução de comandos do OBDH.
- Monitorar o estado do PDC para permitir seu correto funcionamento.
- Coordenar a parada e reiniciação do PDC.

5.4.1 INTERFACES

IObservadorEventos: Interface de observação de eventos. Possibilita um objeto ser notificado da ocorrência de um evento qualquer.

- `PUBLIC notificar(IN evt: Evento): void`
  - **Descrição** - Notifica o observador da ocorrência de evento.
  - **evt** - O evento a ser notificado.
  - **retorno** - Nenhum.

ISujeitoEventos:

- `PUBLIC adicionarObservador(IN obsEvt: IObservadorEventos): void`
  - **Descrição** - Adiciona observador de evento.

- **obsEvt** - Observador de evento a ser adicionado.
- **retorno** - Nenhum.

- `PUBLIC removerObservador(IN obsEvt: IObservadorEventos): void`
  - **Descrição** - Remove observador de evento.
  - **obsEvt** - Observador de evento a ser removido.
  - **retorno** - Nenhum.

5.4.2 CLASSES

AtuarHWCmd: Manipulador do comando de atuação no hardware ATUAR_HARDWARE (0x01).

CarregarDdoCmd: Manipulador do comando que solicita a carga de dados/programas na memória do PDC (*load*) (0x0D).

Comando: Conjunto das operações comuns aos manipuladores de mensagens de comando do OBDH.

- `PUBLIC configurar(IN cmd: MsgOBDH): void`
  - **Descrição** - Ajusta um manipulador de mensagem de comando para executar o comando representado pela mensagem recem chegada.
  - **cmd** - Mensagem de comando do OBDH.
  - **retorno** - Nenhum.
- `PUBLIC executar(): void`
  - **Descrição** - Invoca a execução do manipulador de comando. Eventualmente, o manipulador atuará em diversas partes da aplicação, por meio das diversas interfaces dos componentes, produzindo e enviando uma mensagem de resposta ao OBDH.
  - **retorno** - Nenhum.

EnviarRgoCmd: Manipulador do comando ENVIAR_RELOGIO (0x02). Recupera o tempo atual do sistema e produz uma mensagem de resposta do tipo DADOS_DE_RELOGIO (0x82), solicitando o envio da resposta à classe **ComunicadorOBDH**.

ExecutarPrgCmd: Manipulador do comando que reorganiza a memória do PDC para executar o carregado na página de carga de dados/programas (0x0B).

FabricaComandos: Determina qual manipulador de comando deve ser invocado em função de mensagens de comando do OBDH.

- `PUBLIC obterManipuladorCmd(IN msg: ):`
  - **Descrição** - Obtém um manipulador de comando em função de uma mensagem de comando do OBDH.
  - **msg** - A mensagem de comando do OBDH.

- **retorno** - O manipulador de comando apropriado.

ModificarParSWCmd: Manipulador do comando MODIFICAR_PARAMETROS_SOFTWARE (0x0F). Modifica os parâmetros de software por meio da interface **IConfiguracao**.

MudarModOpeCmd: Manipulador do comando para efetuar a troca de modo de operação (0x03).

NuloCmd: Representa um manipulador de comandos nulo, que não realiza qualquer tarefa. Usado para prevenir erros, caso a fábrica de comandos não encontre um manipulador apropriado.

ObsEvtCfg: Observador de eventos relativos à mudança na configuração do sistema.

ObsEvtComEPP: Observador de eventos na comunicação com o EPP.

ObsEvtComOBDH: Observador de eventos na comunicação com o OBDH.

ObsEvtErrHW: Observador de erros de hardware.

PararAquDdoCmd: Manipulador do comando para interromper a aquisição (0x05) de dados científicos em andamento.

PrepararDgaCmd: Manipulador do comando que prepara um sessão de transmissão de dados de descarredos de uma região específica da memória do PDC (*dump*) (0x0A).

PrepararDgeCmd: Manipulador do comando que prepara um sessão de transmissão de dados de diagnóstico a serem adquiridos junto ao EPP (0x0B).

PrepararHKCmd: Manipulador do comando que prepara um sessão de transmissão de dados de *housekeeping* (0x09).

PrepararTstCmd: Manipulador do comando que prepara uma transmissão de dados de teste a serem adquiridos junto ao EPP (0x0C).

ReiniciarAquDdoCmd: Manipulador do comando para reiniciar a aquisição (0x06) de dados científicos.

ReiniciarCtrErrCmd: Manipulador do comando que efetua o zeramento dos contadores de erros (0x0B).

RetransmitirResposta: Manipulador do comando para retransmitir a última resposta processada de acordo com o tipo de dados solicitado (0x08).

SWPDC: Contém o ponto de entrada principal do programa. Representa a classe que contém o fluxo principal de execução da aplicação. Além disso, coordena a iniciação do SWPDC e ativa todos os serviços (tarefas), viabilizando a execução das funcionalidades de toda a aplicação.

- `PRIVATE configurarCmd(IN MsgOBDH*: ): void`
  - **Descrição** - Configura comando passado como parâmetro.
  - **MsgOBDH*** - Mensagem no Formato de resposta para OBDH

- **retorno** - Nenhum.

- `PRIVATE executar(): void`
  - **Descrição** - Ponto de entrada da terafa principal: obter e processar mensagens de comando, uma vez que o **ComunicadorOBDH** sinalar o recebimento de uma mensagem válida.
  - **retorno** - Nenhum.
- `PRIVATE iniciarObsEvt(): void`
  - **Descrição** - Inicia os observadores de eventos.
  - **retorno** - Nenhum.
- `PRIVATE iniciarRTOS(): void`
  - **Descrição** - Configura e inicia as capacidades de RTOS (*Real-Time Operating System*) adotadas para viabilizar multitarefa no SWPDC.
  - **retorno** - Nenhum.
- `PRIVATE iniciarTarefas(): void`
  - **Descrição** - Inicia tasks do SWPDC.
  - **retorno** - Nenhum.
- `PUBLIC main(): void`
  - **Descrição** - Ponto de entrada principal da aplicação SWPDC.
  - **retorno** - Nenhum.
- `PRIVATE pararTarefas(): void`
  - **Descrição** - Pára as tarefas do SWPDC. Usado durante a reiniciação do sistema.
  - **retorno** - Nenhum.
- `PRIVATE registrarObservadores(): void`
  - **Descrição** - Registra os observadores de eventos em seus respectivos sujeitos.
  - **retorno** - Nenhum.
- `PRIVATE reiniciar(): void`
  - **Descrição** - Causa a reiniciação do sistema.
  - **retorno** - Nenhum.
- `PRIVATE setFlgEppHx1(): void`
  - **Descrição** - Seta o flag indicando que pode começar a enviar comandos para a EPP-HX1 solicitando dados.
  - **retorno** - Nenhum.
- `PRIVATE setFlgEppHx2(): void`

- **Descrição** - Seta o flag indicando que pode começar a enviar comandos para a EPP-HX2 solicitando dados.
- **retorno** - Nenhum.

Atributos da classe **SWPDC**:

- `PRIVATE m_obsEvtComEPP: ObsEvtComEPP`

  O observador de eventos inerentes a comunicação com o EPP.
- `PRIVATE m_obsEvtComOBDH: ObsEvtComOBDH`

  O observador de eventos inerentes a comunicação com o OBDH.
- `PRIVATE m_obsEvtErrCG: ObsEvtErrHW`

  O observador de eventos de erros de cão-de-guarda.
- `PRIVATE m_obsEvtErrHW: ObsEvtErrHW`

  O observador de eventos de erros no hardware.

TransmitirDdoCmd: Manipulador do comando que solicita a transmissão de algum tipo de dado (0x07).

VrfModOpeCmd: Executa o comando de verificação de modo de operação atual do SWPDC (04h).

5.5 SUPORTE

O componente de suporte possui as interfaces e classes necessárias para apoiar os serviços do SWPDC, com relação a iniciação e atuação no hardware.

Responsabilidades principais:

- Executar a iniciação do PDC.
- Monitorar erros de hardware.
- Obter dados dos canais analógicos.
- Configurar dispositivos de E/S.
- Atuar em periféricos (comandos ON/OFF).
- Atuar nas memórias de programa e dados.
- Gerenciar temporização associada ao relógio externo.

5.5.1 INTERFACES

IAtuadorHW: Operações relacionadas com iniciação do sistema.

- `PUBLIC cfgADC(): void`
  - **Descrição** - Configura os parâmetros do conversor AD para atuar na leitura das temperaturas.
  - **retorno** - Nenhum.
- `PUBLIC cfgUART0(IN taxa: TaxaEnum): void`
  - **Descrição** - Configura a UART 0 para atuar em 8 bits, sem paridade, 1 stop bit, a uma dada taxa de bits por segundos, usando TIMER2.
  - **taxa** - Taxa (*baud rate*) em bits por segundos de acordo com a enumeração TaxaEnum.
  - **retorno** - Nenhum.
- `PUBLIC cfgUART1(IN taxa: TaxaEnum): void`
  - **Descrição** - Configura a UART 1 para atuar em 8 bits, sem paridade, 1 stop bit, a uma dada taxa de bits por segundos, usando TIMER1.
  - **taxa** - Taxa (*baud rate*) em bits por segundos de acordo com a enumeração TaxaEnum.
  - **retorno** - Nenhum.
- `PUBLIC desligarEHX1(): void`
  - **Descrição** - Envia o sinal OFF para tentar desligar o conjunto EPP-HX1.
  - **retorno** - Nenhum.
- `PUBLIC desligarEHX2(): void`
  - **Descrição** - Envia o sinal OFF para tentar desligar o conjunto EPP-HX2.
  - **retorno** - Nenhum.
- `PUBLIC iniReg(): void`
  - **Descrição** - Inicia os registradores de mapeamento de portas.
  - **retorno** - Nenhum.
- `PUBLIC ligarEHX1(): void`
  - **Descrição** - Envia o sinal ON para tentar ligar o conjunto EPP-HX1.
  - **retorno** - Nenhum.
- `PUBLIC ligarEHX2(): void`
  - **Descrição** - Envia o sinal ON para tentar ligar o conjunto EPP-HX2.
  - **retorno** - Nenhum.
- `PUBLIC limparFlash(IN endereco: int32u): int8u`

- **Descrição** - Limpa a página da memória flash que contém o endereço passado como parâmetro.
- **endereco** - Endereço de 32 bits que contém informação da página a ser apagada.
- **retorno** - 1 - operação bem sucedida, ou zero caso contrário.

- `PUBLIC obtStaAlmEHX1(): bflag`
  - **Descrição** - Obtém o status da alimentação do conjunto EPP-HX1.
  - **retorno** - 1 - EHX1 ligado (EHX1_ON = 1), ou zero caso EHX1 desligado.
- `PUBLIC obtStaAlmEHX2(): bflag`
  - **Descrição** - Obtém o status da alimentação do conjunto EPP-HX2.
  - **retorno** - 1 - EHX2 ligado (EHX2_ON = 1), ou zero caso EHX2 desligado.
- `PUBLIC obtStaAlmPDC(): bflag`
  - **Descrição** - Obtém o status da alimentação do PDC.
  - **retorno** - 1 - PDC ligado (potência nominal), ou zero caso desligado.
- `PUBLIC obtStaECG(): bflag`
  - **Descrição** - Obtém o status da porta ECG (Erro de Cão-de-Guarda).
  - **retorno** - Status do sinal ECG: 0 (inativo) ou 1 (ativo).
- `PUBLIC obtStaED(): bflag`
  - **Descrição** - Obtém o status da porta ED (Erro Duplo).
  - **retorno** - Status do sinal ED: 0 (inativo) ou 1 (ativo).
- `PUBLIC obtStaES(): bflag`
  - **Descrição** - Obtém o status da porta ES (Erro Simples).
  - **retorno** - Status do sinal ES: 0 (inativo) ou 1 (ativo)
- `PUBLIC obtStaRST(): bflag`
  - **Descrição** - Obtém o status atual do pino de *reset* (RST).
  - **retorno** - Status do sinal RST: 0 (inativo) ou 1 (ativo).
- `PUBLIC obtVlrTta(IN ponto: int8u): int16u`
  - **Descrição** - Obtém o valor da temperatura no termistor num dado ponto (1 ou 2).
  - **ponto** - 1 - para obter a temperatura no ponto um (porta ADC_C1), ou 2 - para obter a temperatura no ponto dois (porta ADC_C2).
  - **retorno** - Valor da temperatura no ponto indicado.
- `PUBLIC vrfCG(): bflag`
  - **Descrição** - Rotina de verificação do funcionamento do cão-de-guarda.

- **retorno** - Resultado da verificação:1 (OK) ou zero (CG não funcionando corretamente.

- `PUBLIC vrfMemDdo(): int8u`
  - **Descrição** - Executa rotina de verificação (contagem) da memória de dados.
  - **retorno** - Valor 0x00 se a memória de dados NÃO pode ser lida completamente, ou 0x01 caso contrário.
- `PUBLIC vrfMemPrg(): int8u`
  - **Descrição** - Executa rotina de verificação da memória de programa (área de código).
  - **retorno** - Valor 0x00 se a memória de programa está inconsistente, ou 0x01 se a memória de programa está consistente.

IHWEmu: Operações relacionadas com circuito auxiliar para simular erro de cão-de-guarda e conseqüente reset do sistema.

- `PUBLIC ligarAuxOutECG(): void`
  - **Descrição** - Liga a saída do circuito auxiliar para erro de cão-de-guarda.
  - **retorno** - Nenhum.
- `PUBLIC ligarAuxOutRST(): void`
  - **Descrição** - Liga a saída do circuito auxiliar para reset.
  - **retorno** - Nenhum.
- `PUBLIC limparAuxOutECG(): void`
  - **Descrição** - Limpa a saída do circuito auxiliar para erro de cão-de-guarda.
  - **retorno** - Nenhum.
- `PUBLIC limparAuxOutRST(): void`
  - **Descrição** - Limpa a saída do circuito auxiliar para reset.
  - **retorno** - Nenhum.
- `PUBLIC vrfAuxInCG(): bflag`
  - **Descrição** - Verifica entrada do circuito auxiliar de cão-de-guarda.
  - **retorno** - Retorna o status da entrada do circuito auxiliar de cão-de-guarda.

IIniciadorHW: Define a interface de iniciação do SWPDC.

- `PUBLIC iniciar(): RelatoPOST*`
  - **Descrição** - Marca o começo do processo de iniciação do SWPDC, coordenando todos os procedimentos de iniciação do sistema.
  - **retorno** - O relato de POST construído durante a iniciação.
- `PUBLIC m_flgOBDH: bflag`

Flag que indica que pode começar a enviar comandos para OBDH.

ITemporizador: Representa um conjunto de operações para manipulação de tempo do sistema (temporização).

- `PUBLIC habilitado(IN idTimer: int8u): int8u`
  - **Descrição** - Verifica se um dado timer está habilitado ou não.
  - **idTimer** - Identificador do timer.
  - **retorno** - 0 - Timer desabilitado; 1 - Timer habilitado.
- `PUBLIC iniciar(IN idTimer: int8u, IN tempo_ms: int32u): void`
  - **Descrição** - Inicia um temporizador especificado por idTimer com o tempo em milissegundos especificado em tempo_ms. A partir de então, o contador de tempo do referido *timer* será habilitado e decrementado a cada milissegundo até zerar. No entanto, o mesmo permanecerá habilitado, até uma nova iniciação seja feita, tanto para reinicia-lo como para desabilita-lo. Uma chamada *ITemporizador::iniciar(0, 0)*, por exemplo, desabilitará o temporizador zero, pois o parâmetro tempo_ms é zero. Uma chamada *ITemporizador::iniciar(2, 500)*, por exemplo, habilitará o temporizador dois com 500 milissegundos.
  - **idTimer** - Indexador do *Timer* a ser iniciado.
  - **tempo_ms** - Tempo em milissegundos.
  - **retorno** - Nenhum.
- `PUBLIC obter(IN idTimer: int8u): int32u`
  - **Descrição** - Obtém o valor do temporizador especificado no parâmetro idTimer.
  - **idTimer** - Indexador do *timer* para obter seu tempo atual.
  - **retorno** - O tempo corrente do *timer* especificado.

Tarefa: Representa a tarefa a ser executada. Ponteiro para função que não retorna valor e não possui lista de parâmetros.

5.5.2 CLASSES

AtuadorHW: Classe que realiza a interface *IAtuadorHW*. Contém funções relacionadas a atuação no hardware do sistema.

Bootloader: Operações de suporte à atualização do firmware do sistema.

Responsabilidades

- Apagar (liberar) área de boot para receber novo código.
- Copiar novo código que foi previamente carregado com sucesso para a área de boot.

- `PUBLIC apagar(): void`
  - **Descrição** - Apaga a área de boot (BOOT_FLASH_ERASE). Requer que a memória flash secundária esteja mapeada no intervalo [8000h, FFFFh].
  - **retorno** - Nenhum.
- `PUBLIC copiar(): void`
  - **Descrição** - Cópia todo o código atualmente em execução para a área de boot. Requer que a memória flash secundária esteja mapeada no intervalo [8000h, FFFFh] e o código atualmente em execução em Fs0 mapeado de [0000h, 7FFF].
  - **retorno** - Nenhum.

HWEmu: Utilitário de apóio para manipulação de aspectos específicos do hardware.

- `PUBLIC HWEmu_PowerOff(): void`
  - **Descrição** - Ponto de entrada da tarefa que monitora o sinal de alimentação do PDC.
  - **retorno** - Nenhum.

IniciadorHW: Classe que realiza a interface *IIniciadorHW*. Realiza os procedimentos de iniciação do sistema bem como a geração dos relatos de eventos provenientes da seqüência de iniciação (POST).

- `PRIVATE limparFlash(): void`
  - **Descrição** - Comanda a limpeza total da memória Flash (de dados) do sistema.
  - **retorno** - Nenhum.
- `PRIVATE reconfigurar(): void`
  - **Descrição** - Reconfigura periféricos acessando os pinos de portas paralelas associadas ao sistema.
  - **retorno** - Nenhum.
- `PRIVATE setFlgOBDH(): void`
  - **Descrição** - Seta flag que indica que pode começar a enviar comandos para OBDH.
  - **retorno** - Nenhum.
- `PRIVATE verificarHW(): void`
  - **Descrição** - Efetua a verificação do hardware durante o POST.
  - **retorno** - Nenhum.

Atributos da classe **IniciadorHW**:

- `PRIVATE m_relPOST: RelatoPOST`

  Dados coletados durante o POST.
- `PRIVATE staMemDdo: int8u`

Indica sucesso ou falha na leitura da memória SRAM para evitar erros simples e duplos.

- `PRIVATE staMemPrg: int8u`

  Indica sucesso ou falha na Verificação da Memória de Programa.

MonitorErrosHW: Tarefa para monitoramento de erros de hardware. Verifica constantemente o status das portas ED, ES, e ECG.

- `PUBLIC executar(): void`
  - **Descrição** - Ponto de entrada de execução da tarefa.
  - **retorno** - Nenhum.
- `PRIVATE monitorarED(): void`
  - **Descrição** - Lê o sinal da porta ED (Erro Duplo) e notifica os observadores caso ED seja igual a 1.
  - **retorno** - Nenhum.
- `PRIVATE monitorarES(): void`
  - **Descrição** - Lê o sinal da porta ES (Erro Simples) e notifica os observadores caso ES seja igual a 1.
  - **retorno** - Nenhum.
- `PRIVATE notificarObservadores(IN evt: Evento): void`
  - **Descrição** - Notifica todos os observadores da lista m_listaObs, invocando o método *notificar()*.
  - **evt** - Contexto da notificação de evento aos observadores.
  - **retorno** - Nenhum.

Atributos da classe **MonitorErrosHW**:

- `PUBLIC m_listaObs: IObservadorEventos`

  Referências para os observadores de eventos interessados em serem avisados quando da ocorrência de erros de hardware. Os erros são: erros simples, erros duplos e de cão-de-guarda não atualizado.

Startup: Representa o código totalmente dependente da plataforma.

Principais resposabilidades:

- Configurar o controlador do barramento externo (BUSCON).
- Reconfigurar o mapeamento de memória, determinando a necessidade de atualização em função de uma eventual carga de novo firmware.
- Se necessário, efetuar a formatação do setores da memória *flash* secundária (CSBOOT) para receber o novo firmware.
- Configurar a pilha de chamadas (registrador SP) para seu valor inicial.

- Invocar o método principal **servicos::SWPDC::main()**.

- `PUBLIC c_start(): void`
  - **Descrição** - Ponto de entrada de iniciação do sistema. Representa o código de iniciação do hardware e preparação do firmware, registrado no primeiro *slot* do vetor de interrupções.
  - **retorno** - Nenhum.

Temporizador: Classe que implementa a interface *ITemporizador*. Os temporizadores (*timers*) são implementados como um vetor de inteiros de 32 bits. Esta implementação oferece apenas 4 temporizadores, além do contador de tempo corrente do sistema.

- `PRIVATE extInt0ISR(): void`
  - **Descrição** - Serviço de tratamento da interrupção externa EXTINT0, associada ao oscilador externo para marcar o tempo em milissegundos. Incrementa o tempo corrente do sistema e decrementa os *timers* não zerados.
  - **retorno** - Nenhum.

Atributos da classe **Temporizador**:

- `PUBLIC m_tempoAtual: int32u`

  Representa tempo em milissegundos passado desde ultima iniciação do sistema.

- `PRIVATE m_timers: TimerInfo`

  Vetor de temporizadores.

TimerInfo: Informações sobre um timer específico.

Atributos da classe **TimerInfo**:

- `PUBLIC habilitado: int8u`

  Flag que indica se o timer está habilitado ou não.

- `PUBLIC tarefa: Tarefa`
- `PUBLIC tempo: int32u`

  Tempo restante em milissegundos.

pdcsys: Definições globais, estruturas de dados, tipos e constantes do sistema.

Atributos da classe **pdcsys**:

- `PUBLIC MAX_COMP_MSG_OBDH:`

  Comprimento máximo em bytes de uma mensagem segundo o protocolo PDC-OBDH.

- `PUBLIC MAX_DADO_MSG_OBDH:`

  Comprimento máximo em bytes do campo de dados de uma mensagem segundo o protocolo PDC-OBDH.

- `PUBLIC MAX_RESERV_MSG_OBDH:`

  Quantidade de bytes reservadas no na mensagem de comando PDC-OBDH.

- `PUBLIC PDC_CFO_PAGE_ID: int`

Identificação da página de dados do buffer de dados científicos.

- `PUBLIC PDC_DGE_PAGE_ID: int`

  Identificação da página de dados do buffer de dados de diagnóstico.

- `PUBLIC PDC_HKE_PAGE_ID: int`

  Identificação da página de dados do buffer de dados de *housekeeping*.

- `PUBLIC PDC_MAX_CFO: int`

  Tamanho máximo em bytes do buffer de dados científico.

- `PUBLIC PDC_MAX_DGE: int`

  Tamanho máximo em byte do buffer de dados de diagnóstico.

- `PUBLIC PDC_MAX_HKE: int`

  Tamanho máximo em byte do buffer de dados de *housekeeping*.

TaxaEnum: Enumeração das taxas de transmissão em bytes por segundo para configuração das portas seriais (UARTs).

- `{t115200bps, t19200bps}`

6 REQUISITOS PARA RASTREABILIDADE DO COMPONENTE

6.1 MATRIZ DE RASTREABILIDADE

Artefato	Requisitos Técnicos Relacionados
COMUNICACAO	
IComunicadorOBDH	ET084.
IDriverUART	ET084, ET085.
IMsgOBDH	ET084.
ComunicadorEPP	ET014, ET015.7, ET015.8, ET040, ET043, ET044, ET085, ET091.
ComunicadorOBDH	ET013, ET015.4, ET015.5, ET039, ET041, ET042, ET058, ET084, ET090.
DriverEPP	ET085.
DriverOBDH	ET084.
MsgOBDH	ET084.
PacoteEPP	ET085.
DADOS	
IBuffer	ET066.
IBufferSimples	ET069.
IBufferVirtual	ET066, ET068, ET070, ET071.
IFormatador	ET068, ET069, ET070, ET071, ET078, ET079.
IGerenciadorDados	ET015.6, ET048, ET053, ET065, ET066, ET079, ET080.
BufferSimples	ET069.
BufferVirtual	ET066, ET068, ET070, ET071, ET079.
Endereco	ET015.6, ET048, ET053, ET065, ET066, ET068, ET070, ET071, ET079, ET080, ET086.
EnderecoFisico	ET015.6, ET048, ET053, ET065, ET066, ET068, ET070, ET071, ET079, ET080, ET086.
Formatador	ET068, ET069, ET070, ET071.
GerenciadorDados	ET015.6, ET048, ET053, ET065, ET066, ET068, ET070, ET071, ET079, ET080, ET086.
Housekeeper	ET016, ET020, ET021,ET022, ET030, ET049, ET060, ET062, ET071, ET086.
MemoriaVirtual	ET015.6, ET048, ET053, ET065, ET066, ET068, ET070, ET071, ET079, ET080, ET086.
PacoteHK	ET020, ET030, ET049, ET062, ET086.
RelatoEvento	ET016, ET086.
Temperatura	ET019, ET032, ET081, ET082.
ESTADO	
IAcaoFsm	ET084, ET085.
ICondicaoFsm	ET084, ET085.
IConfiguracao	ET015, ET016, ET017, ET022, ET057, ET061, ET062, ET081, ET082.
IFSM	ET084, ET085.
Configuracao	ET002, ET022, ET057, ET061, ET062, ET081, ET082.
EstadoFsm	ET084, ET085.
EventoFsm	ET084, ET085.

FSM	ET084, ET085.
MonitorCG	ET035, ET036.
RegraOperacao	ET001, ET002, ET055, ET056, ET061, ET067, ET072, ET073.
TransicaoFsm	ET084, ET085.
SERVICOS	
IObservadorEventos	ET015, ET016, ET086.
ISujeitoEventos	ET015, ET016, ET086.
AtuarHWCmd	ET017, ET026, ET027, ET028.
CarregarDdoCmd	ET061, ET073, ET074.
Comando	ET049, ET058, ET060.
EnviarRgoCmd	ET046.
ExecutarPrgCmd	ET086.
FabricaComandos	ET049, ET059, ET066.
ModificarParSWCmd	ET022.
MudarModOpeCmd	ET018.
NuloCmd	ET086
ObsEvtCfg	ET086.
ObsEvtComEPP	ET016, ET085, ET086.
ObsEvtComOBDH	ET016, ET055, ET067, ET084, ET086.
ObsEvtErrHW	ET016, ET086.
PararAquDdoCmd	ET061.
PrepararDgaCmd	ET063, ET070, ET074, ET079.
PrepararDgeCmd	ET079.
PrepararHKCmd	ET020, ET048, ET049, ET064, ET071, ET075, ET079.
PrepararTstCmd	ET051, ET052, ET069, ET077.
ReiniciarAquDdoCmd	ET061.
ReiniciarCtrErrCmd	ET062, ET086.
RetransmitirResposta	ET048, ET049, ET063, ET064, ET074, ET075, ET077, ET083.
SWPDC	ET001, ET002, ET003, ET004, ET038, ET057.
TransmitirDdoCmd	ET061, ET063, ET064, ET069, ET070, ET074, ET075, ET077, ET078, ET083.
VrfModOpeCmd	ET001, ET002.
SUPORTE	
IAtuadorHW	ET006, ET007, ET008, ET009, ET010, ET011, ET012, ET019, ET023, ET024, ET025, ET026, ET027, ET028, ET030, ET031, ET032, ET033, ET034, ET035, ET036, ET037, ET038, ET039, ET040, ET041, ET042, ET043, ET044.
IHWEmu	ET019, ET023, ET024, ET025, ET026, ET027, ET028, ET030, ET031, ET032, ET033, ET034, ET035, ET036, ET037, ET038, ET039, ET040, ET041, ET042, ET043, ET044.
IIniciadorHW	ET006, ET007, ET008, ET009, ET010, ET057.
ITemporizador	RT045, RT047.
Tarefa	ET094, ET095, ET096, ET097.
AtuadorHW	ET015.1, ET015.2, ET015.3, ET023, ET024, ET025, ET026, ET027, ET028, ET030, ET031, ET032, ET033, ET034, ET035, ET036, ET037, ET038, ET039, ET040, ET041, ET042, ET043, ET044.
Bootloader	ET009, ET010, ET057.
HWEmu	ET019, ET023, ET024, ET025, ET026, ET027, ET028, ET030, ET031, ET032,

	ET033, ET034, ET035, ET036, ET037, ET038, ET039, ET040, ET041, ET042, ET043, ET044.
IniciadorHW	ET006, ET007, ET008, ET009, ET010, ET011, ET012, ET057.
MonitorErrosHW	ET015.
Startup	ET006, ET007, ET008, ET009, ET010.
Temporizador	ET045, ET047.
TimerInfo	ET045, ET047.
pdcsys	ET094, ET095, ET096, ET097.

Os requisitos RT029, RT085, RT094, RT095 e RT096 não foram possíveis de serem mapeados em qualquer artefato do projeto dos componentes.

7 APÊNDICE A: RELAÇÃO DE ACRÔNIMOS PARA FORMAÇÃO DE NOMES DE OBJETOS E VARIÁVEIS

ACRÔNIMO	SIGNIFICADO
ADC	*Analogical/Digital Converter*
AMO	Amostra, amostragem.
AQU	Aquisição
BUFF, BUF	Buffer.
CFO, CCO	Científico.
CG	Cão-de-guarda
CGA	Carga.
COM	Comunicação, comunicador.
CON, CTL	Controlador, controle.
CTR	Contador.
DDO	Dado.
DGE	Diagnose, diagnóstico.
DTA	Data.
END	Endereço.
ENT	Entrada.
EPP	*Event Pre-processor*.
ESP	Espaço.
EVT	Evento.
EXE	Execução, executável, executor.
GER	Gerenciamento, gerenciador.
HK	*Housekeeping, housekeeper*.
HKE	*Housekeeping, housekeeper*.
HRA	Hora, horário.
ID	Identificação, identificador.
IN	*Input*, entrada.
INI	Inicial, início.
MAX	Máximo.
MEM	Memória, endereço ou região de memória,
MIN	Mínimo.
OBDH	*On-board Data-handler Computer*
OBT	Obter, pegar
OUT	*Output*, saída.
PDC	*Payload Data Computer*.
PRG	Programa.
PTO	Ponto.
PTR	Ponteiro.
RTOS	*Real-time Operating System*.
RGO	Relógio.
RX	Recepção de dados.
SAI, OUT	Saída.
SNL	Sinal, sinalizador, sinalização.

SSR	Sensor.
SVC	Serviço, servidor.
TIP	Tipo.
TPO	Tempo, temporização.
TST	Teste.
TTA	Temperatura.
TX	Transmissão de dados.
VLR	Valor

8 APÊNDICE B: PADRÕES DE CODIFICAÇÃO, CONVENÇÕES E PROCEDIMENTOS

8.1 VISÃO GERAL

Esta seção descreve a disciplina e os padrões adotados para mapear os artefatos do projeto físico em código executável na plataforma definida. As seções a seguir descrevem a forma como está estruturada a implementação física software e os padrões de documentação interna do código fonte.

8.2 ESTRUTURA DE DESENVOLVIMENTO

O desenvolvimento do SWPDC contará com um servidor CVS, que viabilizará atividades de controle de configuração, além das estações de trabalho. A Figura 8.1 mostra uma representação da estruturação física do desenvolvimento.

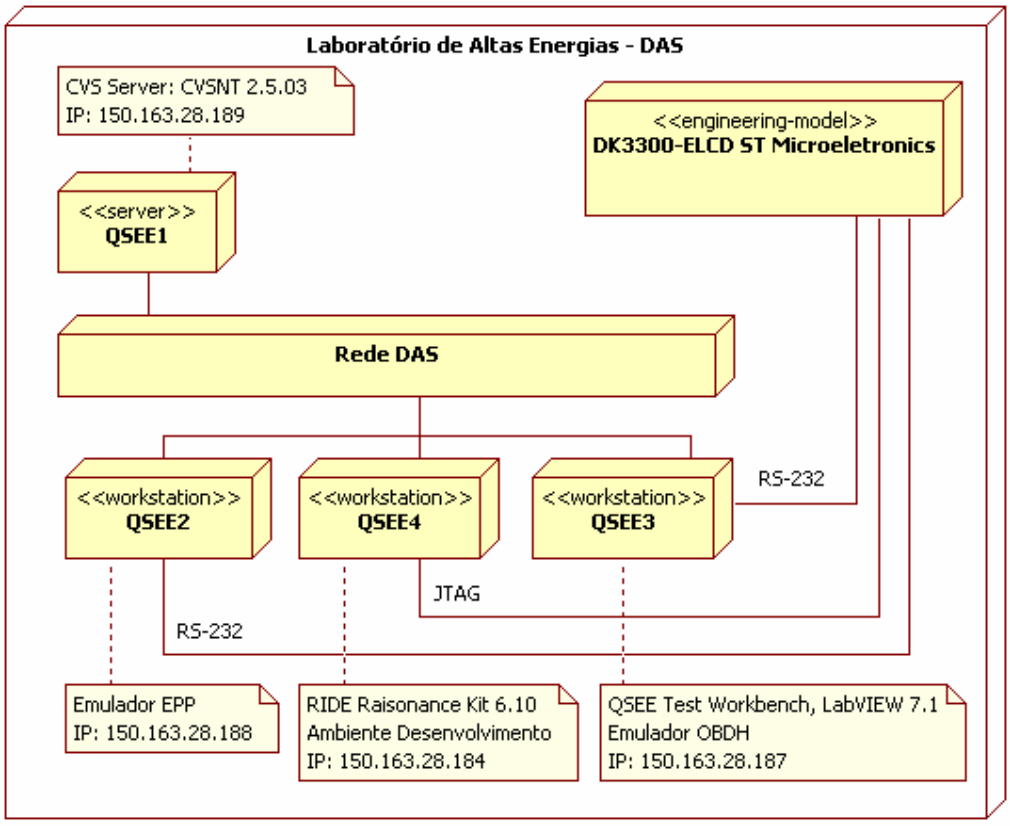

Figura 8.1: Estrutura Fisica de Desenvolvimento

Cada estação de trabalho deve ter um cliente CVS compatível com a versão 2.5.03 do CVSNT, para acessar o repositório qsee1:/cvs.

A estrutura de diretórios (pastas) no repositório qsee1:/cvs é a seguinte:

```
qsee1:/cvs
      |
      |--swpdc-doc [Documentos de projeto do SWPDC]
      |
      `--swpdc-src [Arquivos de código-fonte]
```

O acesso ao repositório poderá ser feito por meio do utilitário de linha de comandos do cliente CVS, ou pelo *WinCVS Client* como mostra o exemplo:

```
cvs -d :pserver:[nome do usuario]@qsee1:/cvs login
```

Uma vez informada a senha do usuário, todas as pastas no repositório poderão ser recuperadas.

A versão mais recente (*snapshot*) de desenvolvimento do SWPDC poderá ser recuperada pelo comando:

```
cvs -d :pserver:[nome do usuario]@qsee1:/cvs chekout swpdc-src
```

A recuperação da versão mais recente (*snapshot*) da documentação do SWPDC pode ser feita pelo comando:

```
cvs -d :pserver:[nome do usuario]@qsee1:/cvs chekout swpdc-doc
```

Durante as atividades de desenvolvimento, sempre antes de uma operação de *commit*, o desenvolvedor deverá executar uma operação de *update*, para se certificar que as mudanças no repositório *cvs*, desde seu último *checkout*, não entrarão em conflito com as modificações efetuadas localmente. Maiores detalhes sobre as melhores práticas com o desenvolvimento apoiado pelo CVS podem ser encontradas em (AD8).

8.3 NOMENCLATURA DE IDENTIFICADORES

O estilo do nome de identificadores deve observar as seguintes recomendações:

1. Nomes de estruturas devem iniciar por letra maiúscula e formam nomes de tipos de dados definidos pelo programador.

2. Nomes de membros de estruturas devem começar com ‘m_’. Exemplo:

```
typedef struct {
    char m_tipo;
    char m_origem;
} Comando;
```

3. Não fazer uso de nomes muito longos para variáveis. Quando necessário, use a concatenação de acrônimos, não excedendo a cinco acrônimos. Para melhor legibilidade, a separação entre acrônimos é feita pela primeira letra do acrônimo em maiúsculo. Por exemplo, para um identificador que represente ‘controlador de

serviços de aplicação', seria codificado por **conSvcApl**. Uma lista de acrônimos é apresentada no Apêndice A.

```
typedef struct {
    ControladorServicos m_conSvcApl;
    GerenciadorBuffer m_gerBuf;
} SWPDC;
```

4. Nomes de tipos que representam objetos do domínio da aplicação devem iniciar por letra maiúscula.

5. Nome de função deve representar ação, portanto, seu nome deve enfatizar verbos.

6. Nome de objeto (ou variável) representa coisas do domínio da aplicação, portanto deve enfatizar substantivos.

7. Identificadores públicos no módulo (.h) ou (.c) devem ser iniciados pela macro PUBLIC.

8. Identificadores locais (ou privados) ao módulo (.h) ou (.c) devem ser iniciados pela macro PRIVATE.

9. As macros PUBLIC e PRIVATE devem ser definidas no arquivo "*pdcsys.h*"

8.4 PADRÕES DE DOCUMENTAÇÃO DO CÓDIGO-FONTE

Os objetivos principais das regras estabelecidas aqui são:

- Geração automática do site de documentação do código-fonte.
- Facilitar a identificação dos artefatos de software e seu conteúdo.

Cada arquivo de código-fonte deve ter um preâmbulo contendo as informações apresentadas na Listagem 8-1.

```
/**
* Instituto Nacional de Pesquisas Espaciais - INPE/SJC/Brasil
* QSEE - Qualidade do Software Embarcado em Aplicações Espaciais
* Software do Computador Gerenciador de Carga Útil - PDC
* Copyright (c) 2005-2006 :: Todos os Direitos Reservados
*
* $Author$
* $Date$
* $Revision$
*
*/
```

Listagem 8-1: Preâmbulo de código-fonte.

As marcações entre os símbolos ($) são palavras-chave do sistema de controle de versões que serão expandidas durante a extração do repositório CVS. Para maiores detalhes, veja a documentação sobre substituição de palavras-chave do CVS em (AD8).

Para documentação interna ao código-fonte, será usado o sistema de documentação *Doxygen*, que usa marcações textuais (*tags*) semelhantes ao sistema Java-Doc. Maiores detalhes, veja documentação em (AD9).

A Listagem 8-2 mostra um exemplo de documentação interna de um arquivo de cabeçalho (.h) que deverá ser observada durante a produção do código-fonte do SWPDC. Note que as palavras-chave do CVS no corpo do preâmbulo encontram-se expandidas.

```
/**
* Instituto Nacional de Pesquisas Espaciais - INPE/SJC/Brasil
* QSEE - Qualidade do Software Embarcado em Aplicações Espaciais
* Software do Computador Gerenciador de Carga de Útil - SWPDC
* Copyright (c) 2005-2006 :: Todos os Direitos Reservados
*
* $Author: wendell $
* $Date: 2006/01/24 10:53:47 $
* $Revision: 1.3 $
*/

/**
 * @file Buffer.h
 * Manipulação de dados em buffer circular.
 */

#ifndef BUFFER_H
#define BUFFER_H
/*****************************************************************************/
/*                              MÓDULOS USADOS                               */
/*****************************************************************************/
#include <stdio.h>
#include <stdef.h>

#include “pdcsys.h”

/*****************************************************************************/
/*                              DEFINIÇÕES E MACROS                           */
/*****************************************************************************/

/**** Limites de quantidade de dados do buffer ****/
#define PDC_MAX_BUFFER_SIZE  1186

/*****************************************************************************/
/*                              TYPEDEFS E ESTRUTURAS DE DADOS               */
/*****************************************************************************/

/**
 * Buffer de dados em memória.
 * Usado para manipular dados (bytes) em memória.
 * Um Buffer não sabe nada a respeito do conteúdo da memória
 * que ele manipula.
 * A estratégia de manipulação dados é do tipo buffer
 * circular (ring-buffer).
 */
struct _Buffer {
```

```
    /**
     * Indexador de leitura do buffer a partir do qual uma
     * unidade pode ser lida.
     */
    unsigned int m_ixrd;
    /**
     * Indexador de escrita do buffer a partir do qual uma unidade
     * pode ser armazenada.
     */
    unsigned int m_ixwr;
    /**
     * Ponteiro de memória a ser manipulada pelo buffer.
     */
    void* m_mem;
    /**
     * Tamanho da memória do buffer; representa o máximo de
     * bytes na memória que o buffer pode manipular.
     */
      size_t m_mem_size;
};

/**
 * Nome de tipo para strutura _Buffer. Definição do tipo Buffer.
 */
typedef struct _Buffer Buffer;

/*****************************************************************************/
/*                          VARIÁVEIS EXPORTADAS                             */
/*****************************************************************************/
#ifndef _BUFFER_C
extern const Buffer TabBuffers[];
extern int8u        NrBuffersAloc;
#endif

/*****************************************************************************/
/*                          FUNÇÕES EXPORTADAS                               */
/*****************************************************************************/

/*****************************************************************************/
PUBLIC int Buffer_adicionar(Buffer* self, void* dado, size_t tamanho_dado);
/*****************************************************************************/
/**
 * @fn int Buffer_adicionar(Buffer* self, void* dado, size_t tamanho_dado);
 * @brief Adiciona dados no buffer.
 * @param self Ponteiro para um buffer.
 * @param dado O dado a ser inserido no buffer.
 * @param tamanho_dado Comprimento em bytes do dado a ser inserido no buffer.
 * @returns Total de bytes efetivamente gravado no buffer.
 */

#endif  //BUFFER_H

/**
 * Histórico de Revisões
 * $Log: Buffer.h,v $
 * Revision 1.3  2006/01/24 10:53:47  wendell
 * Mais exemplos de documentação interna do código-fonte adicionadas.
 *
```

```
 * Revision 1.2  2006/01/24 10:51:29  wendell
 * Declaração da documentação do preâmbulo atualizada.
 */
```

Listagem 8-2: Estilo de documentação interna do código-fonte.

A regra geral é que os comentários de documentação devem estar entre /** e */, e podem ser escritos imediatamente antes das sentenças.

A documentação de funções deve constar de:

- Assinatura da função. [@*fn*]
- Descrição do que ela faz. [@*brief*]
- Descrição sobre o significado de cada parâmetro. [@*param*]
- Descrição sobre o significado do valor de retorno, quando houver. [@*returns*]

Todo arquivo de cabeçalho (arquivo.h) deverá proteger seu conteúdo de re-inclusões, usando instruções de pré-processador como mostra o exemplo abaixo:

```
#ifndef NOME_DO_ARQUIVO_H
#define NOME_DO_ARQUIVO_H
Declarações ...
#endif
```

O símbolo `NOME_DO_ARQUIVO` deve ser grafado em letras maiúsculas.

A documentação interna da implementação deve seguir um padrão semelhante. Maiores detalhes a respeito da documentação do código-fonte, veja os arquivos "*template.h*" e "*template.c*" no repositório de documentação (swpdc-doc). Essas práticas de documentação têm como base de referência (RD3) e (RD6).