Korean WA-DGNSS User Segment Software Design

S.C. Shah, W.S. Choi & W.Y. Han
ICT Convergence Autonomous Research Team, Electronics and Telecommunications Research Institute, Korea

H. Yun & C. Kee
Mechanical and Aerospace Engineering and Institute of Advanced Aerospace Technology, Seoul National University, Korea

ABSTRACT: Korean WA-DGNSS is a large scale research project funded by Ministry of Land, Transport and Maritime Affairs Korea. It aims to augment the Global Navigation Satellite System by broadcasting additional signals from geostationary satellites and providing differential correction messages and integrity data for the GNSS satellites. The project is being carried out by a consortium of universities and research institutes. The research team at Electronics and Telecommunications Research Institute is involved in design and development of data processing softwares for wide area reference station and user segment. This paper focuses on user segment software design. Korean WA-DGNSS user segment software is designed to perform several functions such as calculation of pseudorange, ionosphere and troposphere delays, application of fast and slow correction messages, and data verification. It is based on a layered architecture that provides a model to develop flexible and reusable software and is divided into several independent, interchangeable and reusable components to reduce complexity and maintenance cost. The current version is designed to collect and process GPS and WA-DGNSS data however it is flexible to accommodate future GNSS systems such as GLONASS and Galileo.

1 INTRODUCTION

Korean WA-DGNSS is a large scale research project funded by Ministry of Land, Transport and Maritime Affairs Korea [1] [2]. It aims to augment the Global Navigation Satellite System [4] by broadcasting additional signals from geostationary satellites and providing differential correction messages and integrity data for the GNSS satellites. Korean WA-DGNSS uses a network of wide area reference stations, wide area master station, ground earth station, and geostationary satellites.

Wide area reference stations are widely dispersed GNSS data collection sites that monitor and process satellite data in order to determine satellite orbit and clock drift plus delays caused by the atmosphere and ionosphere. This information is then transmitted to wide area master station through terrestrial communication network. Wide area master station creates correction messages and broadcasts them through geostationary satellites. The user segment receives the correction messages and applies them in order to improve position accuracy and reliability [5].

Korean WA-DGNSS project is being carried out by a consortium of universities and research institutes. The research team at Electronics and Telecommunications Research Institute is involved in design and development of data processing softwares for wide area reference station and user segments. This paper focuses on user segment software design. The user segment software uses GPS data broadcast from each GPS satellite to determine its position and WA-DGNSS corrections messages broadcast from geostationary satellites to improve accuracy and

integrity. It is designed to perform several functions such as calculation of pseudorange, ionosphere and troposphere delays, application of fast and slow corrections messages, integrity monitoring, data verification, and performance visualization.
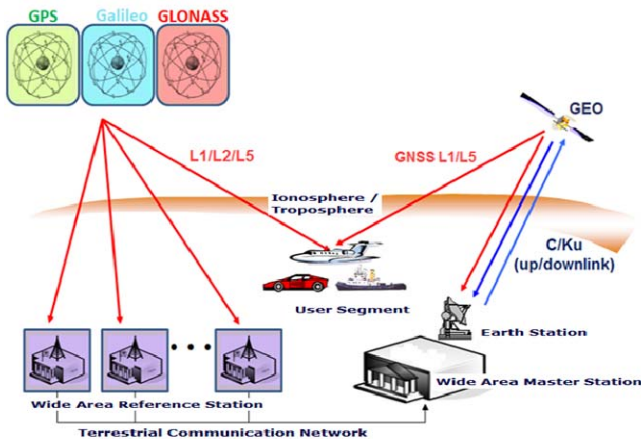

Figure 1. Korean WA-DGNSS Architecture

The user segment software is based on a layered architecture that provides a model to develop flexible and reusable software and is divided into several independent, interchangeable and reusable components to reduce complexity and maintenance cost. The current version is designed to collect and process GPS and WA-DGNSS data however it is flexible to accommodate future global navigation satellite systems such as GLONASS and Galileo.
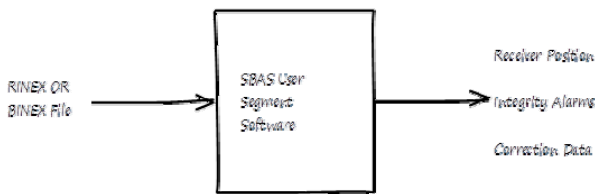

Figure 2. User Segment Software Block Diagram

The rest of the paper is organized as follows. Section 2 provides a high level overview of user segment software architecture whereas Section 3 describes key data processing algorithms. The detailed design of user segment software is given in Section 4. Section 5 discusses the characteristics of User Segment software while Section 6 concludes the paper.

2 ARCHITECTURE DESIGN

A high level overview of user segment software architecture is given in Figure 3. This section briefly describes the key architectural elements. For a detailed description, the readers are referred to User Segment Software Architecture document [6].

2.1 Description of key architectural elements

2.1.1 GPS Ephemeris Data Processor

This component determines the position of GPS satellite with respect to user segment and applies corrections provided by WA-DGNSS. It accesses data from GPS Ephemeris Data Handler, GPS Ephemeris Data Quality Monitor and Slow Corrections Data Processor, and provides an interface to SV Azimuth and Elevation Calculator, GPS SV Clock Data Processor, Receiver Position Calculator, and Raw and Pre-processed Data Buffer.

2.1.2 Slow Corrections Data Processor

This component processes slow corrections messages for GPS ephemeris and clock errors. It accesses data from Slow Corrections Data Handler and provides an interface to GPS Ephemeris Data Processor, GPS SV Clock Data Processor and Raw and Pre-processed Data Buffer.

2.1.3 Independent Data Verifier

This component independently verifies the integrity of active data. It accesses the data from Data Processors, Pseudorange Calculator and Receiver Position Calculators, and provides an interface to Raw and Pre-processed Data Buffer.

2.1.4 GPS SV Clock Data Processor

This component calculates SV clock corrections. It accesses the data from GPS SV Data Handler, GPS SV Data Quality Monitor, GPS Ephemeris Data Processor and Slow Corrections Data Processor, and provides an interface to Pseudorange Calculator.

3 DATA PROCESSING ALGORITHMS

This section describes the key algorithms required to process fast and slow corrections messages. A detailed description of data processing algorithms is available in the SBAS User Segment Software Detailed Design document [7].

3.1 Algorithm to process fast corrections

```
IF (Message Type == 0)
    System Testing
IF (Message Type == 1)
    Save PRN mask and Issue of Data PRN (IODP)
    provided in Message
IF (Message Type == 2-5 OR 24)
    IF (IODP != IODP provided in Message Type 1)
        Return
    IF (UDREI == 14 OR UDREI == 15)
        Initialize RRC calculation and set RRC = 0
        Set satellite status and Return
    IF (PRC is not valid) Return
    IF (precision approach)
        IF (UDREI>=12)
            Satellite status = "do not use" and Return
    IF ( ai_i ==0) Set RRC = 0
    ELSE
        Select previously received pseudorange
```

correction PRC$_{previous}$
IF (IODF of PRC$_{current}$ == 3)
    Select PRC$_{previous}$ which is closest to $I_{fc}/2$
    seconds prior to the PRC$_{current}$
IF (IODF<3)
    Select most recently transmitted PRC$_{previous}$
Calculate range rate corrections
Calculate fast corrections:
Check RRC status
    IF (RRC status is valid)
        Calculate fast corrections
IF (Message Type == 6)
IF (IODF$_j$== 3)
    IF (message is not valid) Return
ELSE
    Decode and apply UDRE to corrections

Recalculate PR
IF (IODF$_j$ < 3)
    Match IODF$_j$ with IODF$_j$ provided in Message
    Type j (2-5 and 24)
    IF (not matched) Return
    IF (message is not valid) Return
    ELSE
        Decode and apply UDRE bounds to corrections
        provided in Message Type j
        Recalculate PR
IF (Message Type == 7)
    IF (IODP != IODP provided in Message Type 1)
        Return
    ELSE
        Store UDRE degradation and time-out intervals
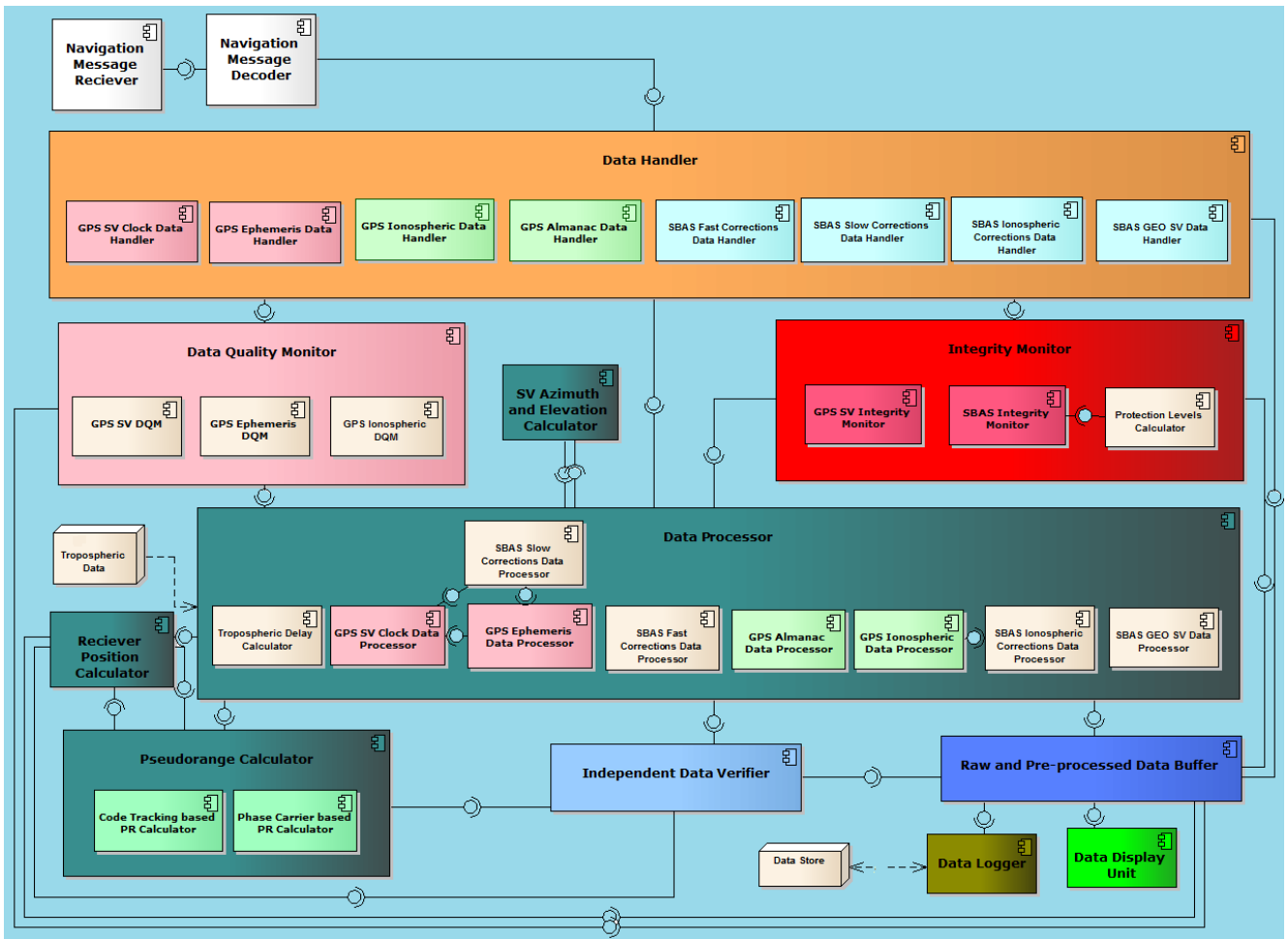
Figure 3. User Segment Software Architecture

3.2 *Algorithm to apply fast and slow corrections*

Integrity Check:
IF (UDREI==15)
    SV is unhealthy
    IF (UDREI==14 OR SBAS data not valid OR IOD! =IODE)
        SV unmonitored
    IF (Precision approach mode)
        IF (UDREI in Message Types 2-5 and 24 >= 12)
            Do not use satellite

IF (Precision approach mode)
    IF (Any of correction messages are missed AND
        previously received corrections are valid)
        Apply degradation models [7]
        Apply fast, ionospheric, tropospheric and clock

corrections:

$$PR = PR_{measured} + FC - IC_{SBAS} + TC_{SBAS} + CC$$

Apply slow corrections:

$$\begin{bmatrix} x_{corrected} \\ y_{corrected} \\ z_{corrected} \end{bmatrix} = \begin{bmatrix} x_{GPS} \\ y_{GPS} \\ z_{GPS} \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix}$$

IF (Correction messages are not missed AND
    previously received corrections are available)
    Apply fast and slow corrections [7]
ELSE
    Precision approach not possible
Exit

IF (Non-precision approach mode)
   IF (Fast and slow corrections are available)
      Apply fast corrections:

$$PR = PR_{measured} + FC + CC$$

      Apply slow corrections:

$$\begin{bmatrix} x_{corrected} \\ y_{corrected} \\ z_{corrected} \end{bmatrix} = \begin{bmatrix} x_{GPS} \\ y_{GPS} \\ z_{GPS} \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix}$$

   IF (SBAS ionospheric corrections are available)
      Apply SBAS ionospheric corrections:

$$PR = PR_{measured} + FC - IC_{SBAS} + CC$$

   ELSE IF (GPS ionospheric corrections available)
      Apply GPS ionospheric corrections:

$$PR = PR_{measured} + FC - IC_{GPS} + CC$$

   ELSE
      Do not apply ionospheric corrections
   IF (SBAS tropospheric corrections are available)
      Apply SBAS tropospheric corrections:

$$PR = PR_{measured} + FC - IC_{SBAS} + TC_{SBAS} + CC$$

   ELSE
      Do not apply tropospheric corrections
ELSE
   Do not use the SV

3.3 *Algorithm to calculate pseudorange*

Pseudorange residual implies the remaining error obtained after eliminating errors such as satellite clock bias, ionospheric delay, tropospheric delay, and receiver clock error from the smoothed pseudorange of satellite j.

   Calculate distance between user and satellite j:

$$D_j = \sqrt{(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2}$$

(x, y, z) User segment coordinates

(x_j, y_j, z_j) Satellite j coordinates

   Calculate Pseudorange:

$$PR_j = D_j + d_{iono} + d_{tropo} - \Delta t_{sv} + B + e_p$$

e_p Measurement noise

B is receiver clock error

   Smooth Pseudorange:

$$SPR_j = \frac{1}{M} \sum_{i=1}^{M} PR_{Cl}$$

PR_{Cl} is available from navigation message

M is moving average points

   Calculate Pseudorange Residuals:

$$PRR_j = SPR_j - D_j - d_{iono} - d_{tropo} + \Delta t_{sv} - B$$
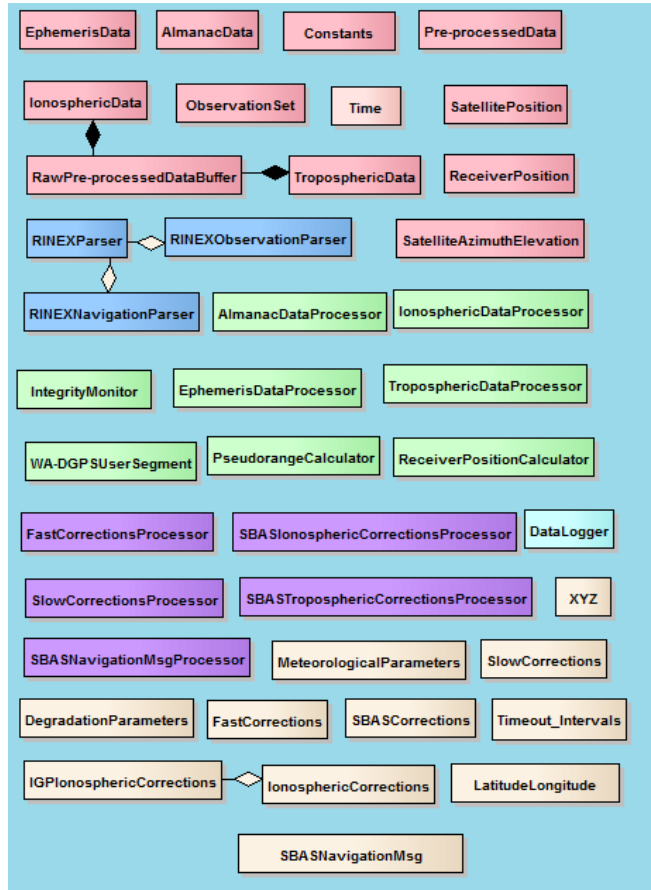
Figure 4. Classes of User Segment Software

4 CLASS DIAGRAMS

The class diagrams are used to describe the static structure of a system [3]. Each class in a class diagram has attributes, operations and the relationship with other classes. The classes are represented by rectangles which show the name of the class and optionally the name of the operations and attributes. Compartments are used to divide the class name, attributes and operations.

   The classes used in WA-DGNSS User Segment software are listed in Figure 4. The classes are divided into five categories, each represented with a different color. The classes in light red color are used to hold raw and pre-processed data whereas classes in light yellow color are used to hold corrections data. Classes in light green and purple colors are data processing classes that are used to process raw GPS data and WA-DGNSS corrections, respectively. The classes in light blue color are used to parse RINEX navigation and observation files. The class diagram of raw GPS data processing classes is depicted in Figure 5. A detailed description of each class is specified in the SBAS User Segment Software Detailed Design document [7].

EphemerisDataProcessor

- argumentLatitudeCorrection :double
- argumentPerigee :double
- correctedArgumentOfLatitude :double
- correctedInclination :double
- correctedMeanMotion :double
- correctedOrbitalRadius :double
- eccentricAnomaly :double
- gpsCorrctedTransmissionTime :double
- inclinationCorrection :double
- longitudeOfAscendingNode :double
- meanAnomaly :double
- meanMotion :double
- mode :bool
- orbitalRadious :double
- orbitalRadiusCorrection :double
- satelliteClockError :double
- satellitePosition :SatellitePosition
- semiMajorAxis :double
- t_k :double
- trueAnomaly :double

+ calculateSatelliteAzimuthElevationAngle(satellitePosition :SatellitePosition, wrsPosition :ReceiverPosition) :SatelliteAzimuthElevation
+ checkGPSTimeForWeekCrossover(time :double) :double
+ computeClockCorrectedGPSTransmissionTime(utcTime :long, satelliteClockError :double, obsPseudorange :double) :double
+ computeEccentricAnomaly(time :double, ephemerisData :EphemerisData) :double
+ determineSatelliteClockError() :double
+ determineSatelliteClockError(ephemerisData :EphemerisData) :double
+ determineSatellitePosition() :SatellitePosition
+ determineSatellitePosition(ephemerisData :EphemerisData) :SatellitePosition
+ isEphemerisDataValid(ephemerisData :EphemerisData) :bool
+ isEphemerisDataValid() :bool

+ EphemerisDataProcessor(ephemerisData :EphemerisData) :void
+ EphemerisDataProcessor() :void

IonosphericDataProcessor

- daulFreqUserMode :bool
- gama :double
- ionoDelay :double
- ionoDelaySD :float
- ionosphericData :IonosphericData
- IPPLatitude :double
- IPPLongitude :double
- observationSet :ObservationSet
- satelliteAzimuthElevation :SatelliteAzimuthElevation
- wrsPosition :ReceiverPosition

+ determineIonosphericCorrections(satelliteAzimuthElevation :SatelliteAzimuthElevation, ionosphericData :IonosphericData) :Pre-processedData
+ determineIonosphericCorrections(satelliteAzimuthElevation :SatelliteAzimuthElevation, observationSet :ObservationSet) :Pre-processedData
+ determineIonosphericCorrections() :Pre-processedData

+ daulFreqUserMode() :bool

+ IonosphericDataProcessor(ionosphericData :IonosphericData) :void
+ IonosphericDataProcessor(ionosphericData :IonosphericData, observationSet :ObservationSet) :void
+ IonosphericDataProcessor(observationSet :ObservationSet) :void
+ IonosphericDataProcessor() :void

PseudorangeCalculator

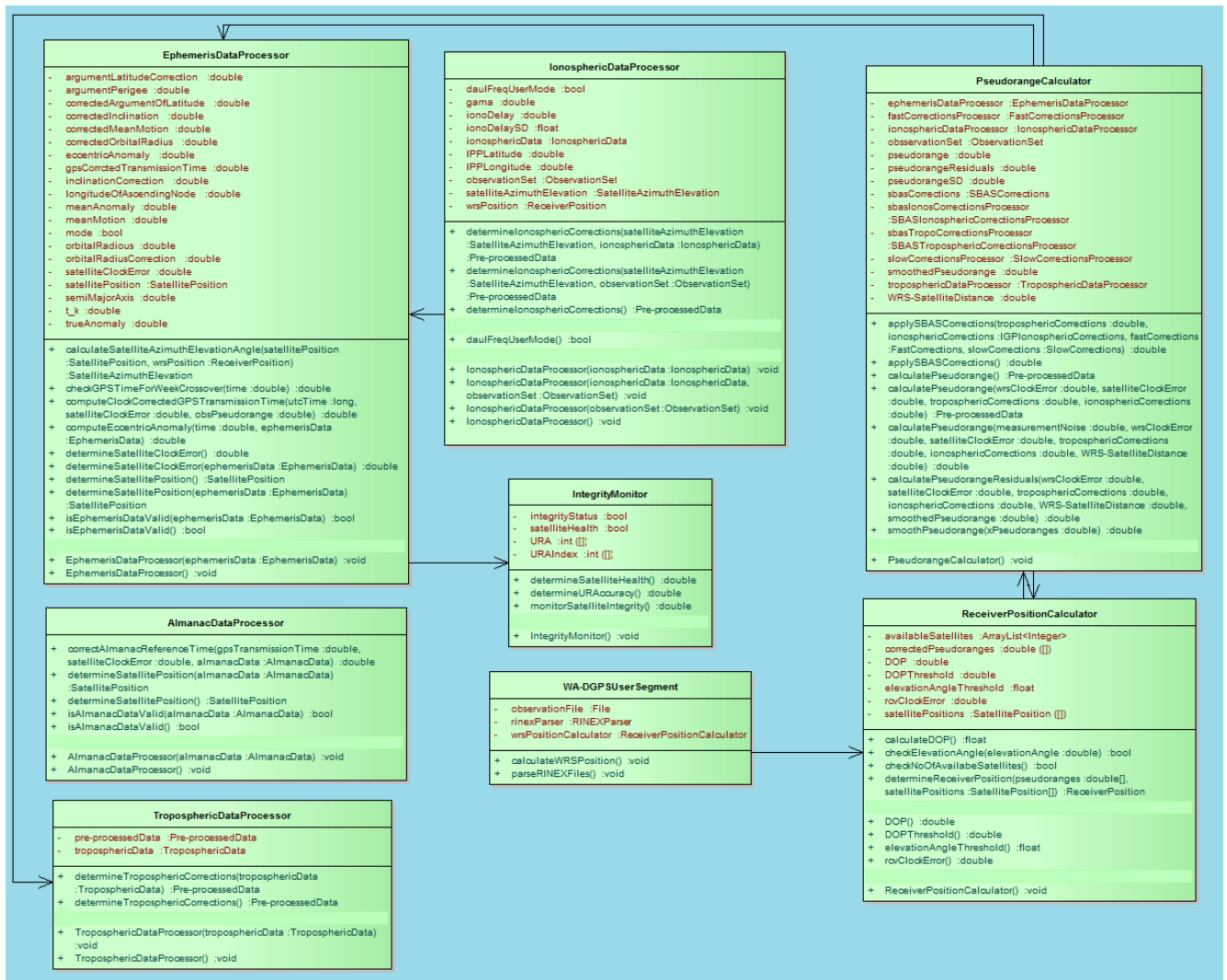- ephemerisDataProcessor :EphemerisDataProcessor
- fastCorrectionsProcessor :FastCorrectionsProcessor
- ionosphericDataProcessor :IonosphericDataProcessor
- obsservationSet :ObservationSet
- pseudorange :double
- pseudorangeResiduals :double
- pseudorangeSD :double
- sbasCorrections :SBASCorrections
- sbasIonosCorrectionsProcessor :SBASIonosphericCorrectionsProcessor
- sbasTropoCorrectionsProcessor :SBASTroposphericCorrectionsProcessor
- slowCorrectionsProcessor :SlowCorrectionsProcessor
- smoothedPseudorange :double
- troposphericDataProcessor :TroposphericDataProcessor
- WRS-SatelliteDistance :double

+ applySBASCorrections(troposphericCorrections :double, ionosphericCorrections :IGPIonosphericCorrections, fastCorrections :FastCorrections, slowCorrections :SlowCorrections) :double
+ applySBASCorrections() :double
+ calculatePseudorange() :Pre-processedData
+ calculatePseudorange(wrsClockError :double, satelliteClockError :double, troposphericCorrections :double, ionosphericCorrections :double) :Pre-processedData
+ calculatePseudorange(measurementNoise :double, wrsClockError :double, satelliteClockError :double, troposphericCorrections :double, ionosphericCorrections :double, WRS-SatelliteDistance :double) :double
+ calculatePseudorangeResiduals(wrsClockError :double, satelliteClockError :double, troposphericCorrections :double, ionosphericCorrections :double, WRS-SatelliteDistance :double, smoothedPseudorange :double) :double
+ smoothPseudorange(xPseudoranges :double) :double

+ PseudorangeCalculator() :void

AlmanacDataProcessor

+ correctAlmanacReferenceTime(gpsTransmissionTime :double, satelliteClockError :double, almanacData :AlmanacData) :double
+ determineSatellitePosition(almanacData :AlmanacData) :SatellitePosition
+ determineSatellitePosition() :SatellitePosition
+ isAlmanacDataValid(almanacData :AlmanacData) :bool
+ isAlmanacDataValid() :bool

+ AlmanacDataProcessor(almanacData :AlmanacData) :void
+ AlmanacDataProcessor() :void

TroposphericDataProcessor

- pre-processedData :Pre-processedData
- troposphericData :TroposphericData

+ determineTroposphericCorrections(troposphericData :TroposphericData) :Pre-processedData
+ determineTroposphericCorrections() :Pre-processedData

+ TroposphericDataProcessor(troposphericData :TroposphericData) :void
+ TroposphericDataProcessor() :void

IntegrityMonitor

- integrityStatus :bool
- satelliteHealth :bool
- URA :int ([])
- URAIndex :int ([])

+ determineSatelliteHealth() :double
+ determineURAccuracy() :double
+ monitorSatelliteIntegrity() :double

+ IntegrityMonitor() :void

WA-DGPSUserSegment

- observationFile :File
- rinexParser :RINEXParser
- wrsPositionCalculator :ReceiverPositionCalculator

+ calculateWRSPosition() :void
+ parseRINEXFiles() :void

ReceiverPositionCalculator

- availableSatellites :ArrayList<Integer>
- correctedPseudoranges :double ([])
- DOP :double
- DOPThreshold :double
- elevationAngleThreshold :float
- rovClockError :double
- satellitePositions :SatellitePosition ([])

+ calculateDOP() :float
+ checkElevationAngle(elevationAngle :double) :bool
+ checkNoOfAvailabeSatellites() :bool
+ determineReceiverPosition(pseudoranges :double[], satellitePositions :SatellitePosition[]) :ReceiverPosition

+ DOP() :double
+ DOPThreshold() :double
+ elevationAngleThreshold() :float
+ rovClockError() :double

+ ReceiverPositionCalculator() :void

Figure 5. Class Diagram of GPS Data Processing Classes

4.1 *EphemerisData Class*

EphemerisData class is used to hold satellite orbit parameters that are used to calculate satellite position and velocity. It is also used to hold parameters for calculating satellite clock corrections. For each parameter, a property is defined that is used to provide a flexible mechanism to read, write, or compute the value of parameter.

4.2 *RawPre-proccssedDataBuffer Class*

A RawPre-proccssedDataBuffer class is a container class that is used to contain objects such as ephemerisData, ionosphericData and observationSet. It also provides methods for adding and removing objects as well as ways to iterate through them.

4.3 *FastCorrections*

FastCorrections class is used to hold fast corrections data such as pseudorange corrections and range rate corrections which are used to remove pseudorange errors. It also provides a mechanism to set and get value of each fast correction.

4.4 *EphemerisDataProcessor Class*

EphemerisDataProcessor class is used to process ephemeris data in order to determine satellite position and clock errors.

4.5 *FastCorrectionsProcessor Class*

FastCorrectionsProcessor class is used to process fast corrections data in order to determine valid pseudorange and range rate corrections.

4.6 *IonosphericDataProcessor Class*

IonosphericDataProcessor class is used to process ionospheric data and observation data in order to determine ionospheric corrections for single and dual frequency users.

4.7 *PseudorangeCalculator Class*

PseudorangeCalculator class is used to calculate pseudorange and pseudorange residuals. The calculated pseudorange includes corrections for ionosphere, troposphere and satellite clock errors.

5 USER SEGMENT SOFTWARE CHARACTERSITICS

5.1 *Modular Software Design*

WA-DGNSS User Segment software is modular in design. It is divided into several independent, interchangeable and reusable components to reduce complexity and maintenance cost. The components can be added or replaced into user segment software without or with minimum affect on the rest of the components. The current software is designed to collect and process GPS and SBAS data however it is flexible to accommodate future GNSS systems such as GLONASS and Galileo.

The new component can be incorporated into user segment software through interfaces that express the elements that are provided and required by the component. If new component adheres to interface, it can be added or replaced without affecting other components.

5.2 *Layered Architecture*

A layer represents a group of related functionality. Layered architecture provides a model to create flexible and reusable software. The user segment software is divided into different layers such as Data Collection and Data Processing where each layer can be easily replaced or new layer can be added without or with minimum affect on other layers.

6 CONCLUSION

This paper describes the detailed design of Korean WA-DGNSS User Segment software in terms of UML architectural, activity and class diagrams. The user segment software is designed to perform several functions such as calculation of pseudorange, ionosphere and troposphere delays, application of fast and slow correction messages, and data verification. It is based on a layered architecture and adopts a modular software design approach.

REFERENCES

[1] Ho Yun, Changdon Kee, Doyoon Kim, Availability Performance Analysis of Korean Wide Area Differential GNSS Test Bed, the Korea Navigation Institute, Vol. 15, No 4, Aug. 2011

[2] Ho Yum, Changdon Kee, and Doyoon Kim, Korean Wide Area Differential Global Positioning System Development Status and Preliminary Test Results, International Journal of Aeronautical and Space Science, Vol. 12, No. 3, 2011, pp. 225-233

[3] Alan Dennis, Barbara Haley Wixom and David Tegarden, Systems Analysis and Design with UML, 4th Edition, Wiley, February 1, 2012.

[4] RTCA SC-159, Minimum Operational Performance Standards for Global Positioning System/Wide Area Augmentation System Airborne Equipment, RTCA/DO-229C, November 28, 2001.

[5] FAA Document, Specification of the Wide Area Augmentation System (WAAS), FAA-E-2892B, 1999

[6] Sayed Chhattan Shah, Wan Sik Choi, Han Woo Yong, Modular Software Architecture for Korean SBAS User Segment, Electronics and Telecommunications Research Institute, Report No. 6060-2012-0439, 2012

[7] Sayed Chhattan Shah, Wan Sik Choi, Han Woo Yong, Detailed Design of Korean SBAS User Segment Software, Electronics and Telecommunications Research Institute, Report No. 6060-2012-042, 2012