Advanced Measurement Infrastructure

(AMI)

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Authors | Date | Comments |
| 1.0 | User | 18.12.2017. | *Initial version, References update, etc.* |

| Responsibility | Role | Name & Surname |
| --- | --- | --- |
| Chief of Architects |  |  |
| QA |  |  |
| Deployment |  |  |
| System/SW Architect |  |  |
| Release Manager |  |  |
| Test manager |  |  |
| Technical Leader |  |  |
| Solution Architect |  |  |

Final review and approval of the latest version:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Date | Name & Surname | Business Role | Signature |
| Reviewer |  |  |  |  |
| Approver |  |  |  |  |

Sadržaj

[1. Definitions and Abbreviations 2](#_Toc441752415)

[2. References 2](#_Toc441752416)

[3. Abstract 2](#_Toc441752417)

[4. Overview 2](#_Toc441752418)

[4.1. Functional Design 2](#_Toc441752419)

[4.2. Architectural Design 2](#_Toc441752420)

[5. Detailed Design 2](#_Toc441752421)

[5.1. Component Design 2](#_Toc441752422)

[5.2. Data Design 2](#_Toc441752423)

[5.3. UI Design 2](#_Toc441752424)

[6. Additional Considerations 2](#_Toc441752425)

[6.1. Quality and Testing 2](#_Toc441752426)

[6.2. Deployment and Configuration 2](#_Toc441752427)

[6.3. Upgradability 2](#_Toc441752428)

[6.4. Extensibility 2](#_Toc441752429)

[6.5. Security 2](#_Toc441752430)

[7. Appendix 2](#_Toc441752431)

# Definicije i skraćenice

|  |  |
| --- | --- |
| Definition/Abbreviation | Description |
|  |  |
|  |  |

# Reference

|  |  |  |
| --- | --- | --- |
| # | Title | Description |
| 1. |  |  |
|  |  |  |

# Uvod

# Pregled

<This should be no more than 1-2 pages where the high level requirements that this architectural document addresses are stated (with the link to the detailed requirements document/user story if exists) and where the high level design is introduced. One global diagram of the relevant architecture is almost a must in this section.>

## Functional Design

<High level description of the functionality – consult relevant Product Owner and/or Business Analyst.>

## Architectural Design

<High level design with main modules, interfaces between them and to the outside world plus main data flows.>

# Detaljan dizajn

## Komponente sistema

Slika 5.1 Prikaz komponenti sistema

Sistem se sastoji iz sledećih komponenti:

* Transaction Coordinator
* Network Model Service (NMS)
* Supervisory Control and Data Acquisition (SCADA)
* Calculation Engine (CE)
* RTU Simulator
* User Interface (UI)

### Transaction Coordinator

***Definicija***

*Transaction Coordinator* je komponenta koja rukuje statičkim podacima, tačnije deltom. Delta predstavlja dokument napisan kao CIM/XML fajl. Uvođenjem ove komponente omogućena je nezavisnost između servisa, kao što su NMS, SCADA i CE. Vrši prosleđivanje delte NMS-u i merenja SCADA sistemu.

***Odgovornosti***

*Transaction Coordinator* obezbeđuje uspešno prosleđivanje podataka servisima. Ukoliko samo jedan od pomenutih servisa nije aktivan delta se neće primeniti.

***Ograničenja***

***Sastav***

*Transaction Coordinator* predstavlja implementaciju *ITransactionCoordinator* interface-a, gde se vrši prosleđivanje podataka. Implementirani su i *ITransactionDuplexNMS* i *ITransactionDuplexScada* na osnovu kojih se SCADA i NMS servisi pretplaćuju na komponentu.

***Interakcija***

Kao što je već rečeno, komponenta komunicira sa SCADA i NMS servisima. Komunikacija je urađena pomoću *duplex* kanala. Kako bi pomenuti servisi dobili njima neophodne podatke, potrebno je prvo da se pretplate na *Transaction Coordinator* servis. Nakon toga, servis prosleđuje podatke pretplaćenim korisnicima ili odbija transakciju ukoliko neko od servisa nije podignut.

***Interface***

*Transaction Coordinator* implementira tri interface.

Prva dva predstavljaju pomenutu pretplatu SCADA i NMS sistema.

|  |
| --- |
| [ServiceContract(CallbackContract = typeof(IScada))]  public interface ITransactionDuplexScada  {  [OperationContract]  void ConnectScada();  } |
| [ServiceContract(CallbackContract = typeof(INetworkModel))]  public interface ITransactionDuplexNMS  {  [OperationContract]  void ConnectNMS();  } |

Treći interface sadrži funkciju u kojoj se vrši prosleđivanje podataka

|  |
| --- |
| [ServiceContract]  public interface ITransactionCoordinator  {  [OperationContract]  bool ApplyDelta(Delta delta);  } |

### Network Model Service – NMS

***Definicija***

***Odgovornosti***

***Ograničenja***

***Sastav***

***Interakcija***

***Interface***

### Supervisory Control and Data Acquisition – SCADA

***Definicija***

SCADA predstavlja sistem za merenje, praćenje i kontrolu industrijskih sistema. Preko *Transaction Coordinator* komponente dobija statičke podatke, dok RTU simulator na svake tri sekunde šalje izmerene vrednosti SCADA sistemu i te vrednosti predstavljaju dinamičke podatke. Nakon toga, SCADA prosleđuje izmerene vrednosti *Calculation Engine* komponenti.

***Odgovornosti***

Može se videti da SCADA sistem rukuje kako statičkim tako i dinamičkim podacima. Tačnije, nakon što dobije statičke podatke pravi kopiju istih i rukuje sa njima. Na osnovu njih javlja RTU simulatoru koliko je potrebno da instalira ‘tačaka’ u svom sistemu i pošalje njihove trenutne vrednosti. Kako bi komponenta mogla da rukuje dinamičkim podacima kreiran je *DynamicMeasurement* u kome će se zapisivati trenutno izmerene vrednosti dobivene od RTU simulatora.

***Ograničenja***

***Sastav***

SCADA sistem implementira *IScada* interface koji obrađuje statičke podatke. Klasa *SOEHandler* zadužena je za obradu dinamičkih podataka.

***Interakcija***

Pri startovanju sistema, SCADA se pretplaćuje na *Transaction Coordinator* kako bi dobila statičke podatke, što je pomenuto u odeljku 5.1.1. Sledeća komunikacija ostvaruje se između SCADE i RTU simulatora. Komunikacija je odrađena preko *duplex* kanala, jer je pre svega potrebno da se RTU simulatori pretplate na SCADU a nakon toga SCADA šalje informacije. Takođe, njihova veza urađena je pomoću DNP3 protokola. Poslednja komunikacija je sa *Calculation Engine* komponentom. Kako SCADA samo prosleđuje izmerene vrednosti CE komponenti, komunikacija je urađena u jednom smeru.

***Interface***

Prvi interface koji SCADA implementira odnosi se na vezu sa *Transaction Coordinator* komponentom.

|  |
| --- |
| [ServiceContract]  public interface IScada  {  [OperationContract]  void EnlistMeas(List<ResourceDescription> measurements);  [OperationContract]  bool Prepare();  [OperationContract]  void Commit();  [OperationContract]  void Rollback();  } |

Drugi interface odnosi se na pretplatu RTU simulatora

|  |
| --- |
| [ServiceContract(CallbackContract = typeof(ISimulator))]  public interface IScadaDuplexSimulator  {  [OperationContract]  void Connect();  [OperationContract]  int GetNumberOfPoints();  } |

### Calculation Engine – CE

***Definicija***

*Calculation Engine* servis predstavlja posrednika između SCADA i UI komponente. Njegova osnovna namera je da podatke koje su primljene od SCADE prosledi pretplaćenim korisnicima.

***Odgovornosti***

***Ograničenja***

***Sastav***

*Calculation Engine* čini implementacija *ICalculationEngine* interface u kome se nalazi funkcija za prosleđivanje podataka.

***Interakcija***

Iz definicije može se zaključiti da CE komunicira sa dve komponente. Veza SCADA – CE implementirana je u jednom smeru kao što je već pomenuto u odeljku 5.1.3. Kako CE prosleđuje podatke samo pretplaćenim korisnicima, njihova komunikacija je urađena pomoću *duplex* kanala, gde je potrebno da se prvo izvrši pretplata klijenata a nakon toga CE šalje podatke.

***Interface***

Prvi interface predstavlja vezu sa SCADA komponentom, dok drugi predstavlja pretplatu korisnika.

|  |
| --- |
| [ServiceContract]  public interface ICalculationEngine  {  [OperationContract]  void DataFromScada(List<DynamicMeasurement> measurements);  } |
| [ServiceContract(CallbackContract = typeof(IModelForDuplex))]  public interface ICalculationDuplexClient  {  [OperationContract]  void ConncetClient();  } |

### RTU Simulator

***Definicija***

DNP3 protokol je skup komunikacionih protokola koji se koriste između komponenti u sistemima automatizacije procesa. Razvijen je za komunikaciju između različitih vrsta opreme za prikupljanje i kontrolu podataka. Prvenstveno se koristi za komunikaciju između glavne stanice i RTU-ova. Glavna stanica, tačnije *master* strana, je SCADA komponenta, dok je RTU *slave* strana.

***Odgovornosti***

RTU simulator je namenjen da navodno prikuplja podatke i šalje ih *master* strani preko dnp3 protokola.

***Ograničenja***

***Sastav***

RTU simulator se sastoji od implementacije interface u kome dobija informaciju od SCADA komponente koliko ‘tačaka’ je potrebno da instalira.

***Interakcija***

RTU simulator komunicira jedino sa SCADA komponentom i to preko *duplex* kanala jer je potrebno da se pre svega simulator pretplati na SCADA sistem.

***Interface***

Sadrži samo jedan interface koji ga povezuje sa SCADA sistemom.

|  |
| --- |
| [ServiceContract]  public interface ISimulator  {  [OperationContract]  int AddMeasurement();  [OperationContract]  void Rollback(int decrease);  } |

### User Interface – UI

***Definicija***

Komponenta omogućava učitavanje i primenu delte tako što šalje deltu *Transaction Coordinator*-u kako bi se ona mogla primeniti. Takođe omogućava prikaz podataka.

***Odgovornosti***

UI komponenta omogućava korisniku dve stvari. Korisnik učitava CIM/XML fajl i šalje zahtev za primenu delte. Na osnovu regiona, subregiona, trafostanica korisnik može da dobije prikaz potrošača kao i vrednosti potrošnje.

***Ograničenja***

***Sastav***

UI komponenta sastoji se od glavnog prozora, *MainWindow*, koji pruža korisniku pomenute dve mogućnosti. Obe ove mogućnosti implementirane su takođe pomoću View komponenti. *AddCimXmlControl* view komponenta služi za učitavanje CIM/XML fajla i zahtev da se primeni delta. *NetworkPreviewControl* view komponenta omogućava prikaz potrošača i potrošnje.

***Interakcija***

UI komponenta komunicira sa NMS-om i CE-om. Komunikacija sa CE-om je već pomenuta u odeljku 5.1.4. Kako bi UI komponenta omogućila prikaz potrošača preko regiona, subregiona i trafostanica potrebno je obezbediti vezu sa NMS-om kako bi dobila odgovarajuće podatke. Veza je urađena preko *duplex* kanala, gde se prvo korisnici pretplaćuju na NMS a nakon toga šalju upite.

***Interface***

UI komponenta sadrži dva interface-a.

Prvi inteface daje informacije koje dobija od CE i NMS komponente.

|  |
| --- |
| [ServiceContract]  public interface IModelForDuplex  {  [OperationContract]  void NewDeltaApplied();  [OperationContract]  void SendMeasurements(List<DynamicMeasurement> measurements);  } |

Drugi interface predstavlja upite koje UI komponenta upućuje NMS-u.

|  |
| --- |
| public interface Imodel  {  ObservableCollection<GeographicalRegion> GetAllRegions();  ObservableCollection<SubGeographicalRegion> GetAllSubRegions();  ObservableCollection<Substation> GetAllSubstations();  ObservableCollection<EnergyConsumer> GetAllAmis();  ObservableCollection<SubGeographicalRegion> GetSomeSubregions(long regionId);  ObservableCollection<Substation> GetSomeSubstations(long subRegionId);  ObservableCollection<EnergyConsumer> GetSomeAmis(long substationId);  void SetRoot(RootElement root);  } |

## Data Design

<Define the data model (examples, choose what is more appropriate: object model diagram, relational model diagram). Describe the persistence model (how is data stored if at all).>

## UI Design

<Describe the User Interface design. Provide mock-ups where appropriate (engage UI designers).>

# Additional Considerations

<Every design document should cover the following areas: Quality and testing, Deployment, Configuration, Upgradability, Extensibility and Security. Generalized statement/paragraph is expected in the body of this chapter (in this exact area) while the details on each subject are to be given in the sub chapters below.>

## Quality and Testing

<This sub chapter should give answers to the following questions:

What existing features are affected by changes (regression area)? What should the test engineer pay attention to when creating test suite and individual test cases? Are there any aspects where extra care is needed when devising tests?

What should be the integration testing strategy? What should be the test bed? Are there any existing tools that can aid in testing, for example to create inputs? Should some tools be developed for this purpose? Recommended minimal system for integration testing should be given.

Are there any special HW/SW/Environment requirements for integration and system testing (having in mind both types of deployment: MyADMS (MyAGMS) and Enterprise)? List those and provide alternatives if some are impossible or hard to attain.

What are the expected non-functional requirements (e.g. performance, availability and sizing requirements) for newly added and/or changed features/components? How are performances of existing features impacted?>

## Deployment and Configuration

<This sub chapter should give answers to the following questions:

Does this design imply changes in the configuration of ADMS (AGMS)? List those and if possible suggest default values.

Is installation packaging affected by this design? If yes, what needs to change?

Are there any post install steps that need to be documented and exercised after each installation? List those and provide explanation.

All questions should be considered and answered for both MyADMS (MyAGMS) and Enterprise editions of ADMS (AGMS).>

## Upgradability

<This sub chapter should assess the impact of this design on the ability to upgrade existing ADMS (AGMS) systems. Following questions should be answered:

Is there any impact on the upgradability of ADMS (AGMS)? If yes, explain what impacts are there and explain how they are addressed.

Please note that the upgradability is affected if one of the following is being done:

- Interfaces declared public are being changed (are we still supporting the old interfaces?)

- Data model which is exposed through public interfaces is changed (are we sill supporting old data model as input and are we providing it as output on the legacy interfaces?)

- Configuration changes have been made (are they addressed during the install/upgrade procedures?)

- Persistence store model (smart caches, SQL databases, flat files …) has changed (are we providing conversion methods? How?)>

## Extensibility

<This sub chapter should give answers to the following questions:

Have considerations been made to the extensibility of the software modules that are subject of this design? Public interfaces including data model have to be identified and then documented during the development process.

Connected to the upgradability sub chapter above, have existing interfaces been changed and are we keeping backwards compatibility?>

## Security

<This sub chapter should address security aspects of the design if any in coordination with software security engineer/team. Following questions should be answered accompanied with the threat model where appropriate:

What are the security risks identified with this feature’s requirements? How the security risk should be managed: reduced, transferred, avoided or accepted?

What threats have been identified? What are the mitigation techniques that should be applied? What impact the threats have on the overall security and what is the probability of those threats being realized?

How to deploy the feature or function in a secure fashion? What application, operating system and network configuration needs to be in place? E.g.:

• Is there a specific firewall rule that needs to be added?

Are there any security rights (group policies) that need to be associated with the user or computer running the feature?>

# Appendix