Katholieke
Universiteit
Leuven

**Department of
Computer Science**

DOCUMENT PROCESSING

The complete architecture

Software Architecture (H09B5a and H07Z9a) – Part 2b

Jeroen Reinenbergh (r0460600)
Jonas Schouterden (r0260385)

Academic year 2014–2015

Contents

1 Introduction

The goal of this project was to develop an architecture for a system for document processing. This part of the project consisted of

2 Overview

2.1 Architectural decisions

Briefly discuss your architectural decisions for each non-functional requirement. Pay attention to the solutions that you employed (in your own terms or using tactics and/or patterns).

ReqX: requirement name Provide a brief discussion of the decisions related to *ReqX*.
Employed tactics and patterns: List all patterns and tactics used to achieve ReqX, if any.

2.2 Discussion

Use this section to discuss your architecture in retrospect. For example, what are the strong points of your architecture? What are the weak points? Is there anything you would have done otherwise with your current experience? Are there any remarks about the architecture that you would give to your customers? Etc.

3 Attribute-driven design documentation

3.1 Decomposition 1: eDocs (X1, Y3, UCa, UCb, UCc)

3.1.1 Module to decompose

In the first run, the eDocs System is decomposed as a whole

3.1.2 Selected architectural drivers

The non-functional drivers for this decomposition are:

- *X1*: name
- *Y3*: name

The related functional drivers are:

- *UCa*: name
- *UCb*: name
- *UCc*: name

Rationale A short discussion of why these drivers were selected for this decomposition.

3.1.3 Architectural design

Topic Discussion of the solution selected for (a part of) one of the architectural drivers.

Alternatives considered

Alternatives for solution A discussion of the alternative solutions and why that were not selected.

3.1.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting decomposition.

ModuleB Per introduced component a paragraph describing its responsibilities.

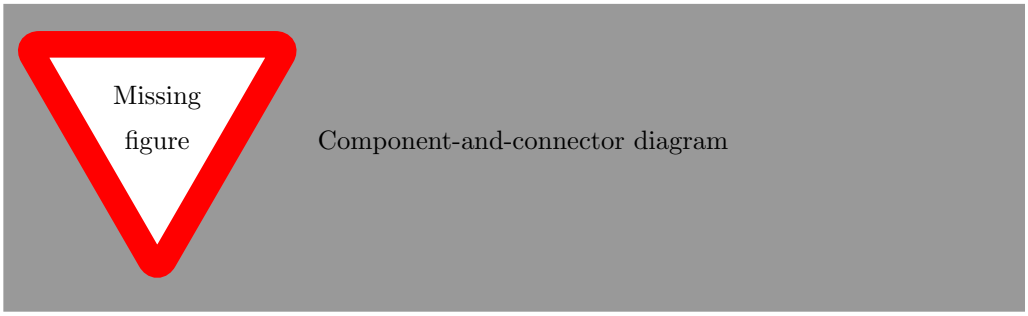


Figure 1: Component-and-connector diagram of this decomposition.

ModuleC Per introduced component a paragraph describing its responsibilities.

Behaviour If needed and explanation of the behaviour of certain aspects of the design so far.

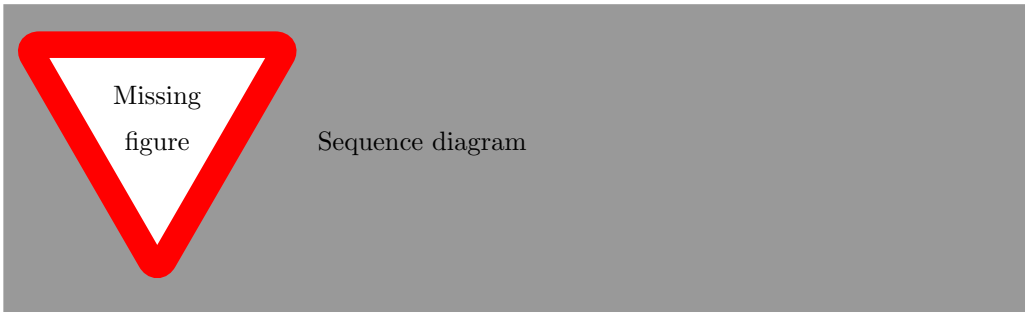


Figure 2: Sequence diagram illustrating a key behavioural aspect.

Deployment Rationale of the allocation of components to physical nodes.

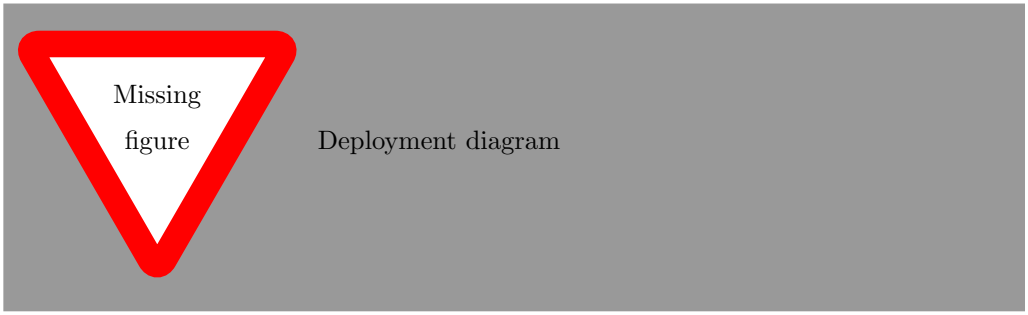


Figure 3: Deployment diagram of this decomposition.

3.1.5 Interfaces for child modules

PDSDB

- DocumentMgmt
  - `void storeDocument(DocumentId id, Document doc, MetaData md)`
    - Effect: The `PDSDB` will store the given document`doc` together with the provided metadata `md`.
    - Exceptions: None
  - `Tuple<Document, MetaData> getDocument(DocumentId id)`
    - Effect: Describe the effect of calling this operation.

    - Exceptions: None
- `void markReceived(DocumentId id)`
    - Effect: Describe the effect of calling this operation.
    - Exceptions: None

3.1.6 Data type definitions

Describe per complex data type used in the interfaces what it represents.

returnType This data element represents X.

ParamType This data element represents Y.

3.1.7 Verify and refine

This section describes per component which (parts of) the remaining requirements it is responsible for.

ModuleB

- *Z1*: name
- *UCd*: name

ModuleC

- *UCba*: name
  Description which part of the original use case is the responsibility of this component.

3.2 Decomposition 2: OtherFunctionality(P2, UC12, UC13, UC14, UC15)

3.2.1 Module to decompose

In this run we decompose `Otherfunctionality`.

3.2.2 Selected architectural drivers

The non-functional drivers for this decomposition are:

- *X1*: name
- *Y3*: name

The related functional drivers are:

- *UCa*: name
- *UCb*: name
- *UCc*: name

Rationale A short discussion of why these drivers were selected for this decomposition.

3.2.3 Architectural design

Extended functionality of PDSLongTermDocumentManager for P2 Discussion of the solution selected for (a part of) one of the architectural drivers.

Separate document and link mapping database for P2

Sharding for document database for P2 Note that there must be a (sub)component monitoring the requests to the different shards, to cap the number of requests. We want to have the response time of active replication. But actively replicating the whole database might have to high a cost, so we choose sharding. This keeps the fast response time, but has less hardware required.

The shards themselves are

Pingen is nodig voor write bij sharding. DocumentStorageManager nodig want twee opslagkanalen, de PDSDB en de DocumentDB

extra methode in PDSDBMngmt voor de opslag

aparte linkdatabase voor P, want "should not affect the performance of other functionality of the system"

UserFacade for P2 we moeten de duplicatie van UserFacade bespreken voor P 2. De userfacade is toegevoegd om onderscheid te maken tussen Registered Recipients en Unregistered Recipients. De userfacade markeert ook documenten als received. De PDSFacade en LinkManager DOEN DIT NIET. Voor r

Dit zowel voor documenten die opgevraagd worden met behulp van een unieke link in een email (voor registered recipients EN voor unregistered recipients) als voor dvia de userfacade. De reden dat we dat op deze plaats doen, en niet bij PDSFacade of LinkManager, is omdat de userfacade de laatste compomnent is waar het document voorbij komt voor he bij de recipient terecht komt. Als we dit in een eerdere component zouden doen, is het mogelijk dat een component dichter bij de recipient faalt zonder dat de gebruiker het document ontvangt en zonder dat dit opgemerkt wordt.

DocumentStorageManager is toegevoegd om de nieuwe componenten te koppelen met de vorige decompositie. De component synchroniseert de DocumentDB met de PDSDB. Hiermee wordt bedoeld dat documenten bedoeld voor Registered Recipients in beide databases worden opgeslagen.

Motivatie voor AuthN: je moet ingelogd zijn voor o.a. use case 12. Motivatie voor UniqueLinkMgmt: het opvragen van documenten met behulp van een unieke link. Motivatie voor PDSDBMgmt: use case 12: the registered recipient indicates that he or she wants to consult his or her personal document store.

Motivatie voor PDSFacade: door een aparte component moet de pdsdb zich net bezighouden met het aggregeren van tussenresultaten bij het verwerken van queries. De PDSFacade wordt niet gerepliceerd, aangezien de PDSDBLongTermStorageManager ook niet gerepliceerd is, en op zich een bottleneck is. De facade is enkel voor de reads, bij het opslaan van documenten gaat de DocumentStorageManager rechtstreek naar de PDSDB. De PDSFacade verzamelt altijd eerst een overview van all the recipients documents DESCRIPTIONS. Subsequently, queries can be performed on this collection. This approach introduces no significant overhead.

Motivatie voor LinkManager: Checks expiration date ALS DAT NODIG IS–¿ reason: links naar de pdsdb vervallen niet (zolang de gebruik geregistreerd is) LinkManager maps link to (document ID, place where the document is stored)-pairs –¿ REASON: the unique link has two possible sources: an e-mail to an unregistered recipient or an email to a registered recipient. For an unregistered recipient, the RecipientFacade must look with the documentid for the document in the documentDB. For a registered recipient, the RecipientFacade must look with the documentid for the document in the PDSDB. (Mogelijk een boolean ofzo)

Does NOT do mapping removal after x years –¿ there has to be a notification when the link has expired

QAS AV2b: hier voldaan, want de documenten worden zowel in de documentdb als in de pdsdb opgeslagen, dus de documenten worden wel nog opgeslagen mocht de pdsdb uitvallen. Een clear message aan de recipient wordt gegeven met behulp van de NotifyRecipient interface aan otherfunctionality2. We hebben twee alternatieve manieren besproken om de drie uur aan documenten op te vangen die mogelijk verloren gaan bij het uitvallen van de PDSDB. De eerste maakt gebruikt van een cache. Die cache slaat de documentids en de recipient-ids op die de laatste drie uur gegenereerd zijn. Aangezien de documenten die in de PDSDB horen opgeslagen te worden ook in de documentDB zitten, kunnen dan de documenten van de laatste drie uur door de DocumentStorageManager uit de DocumentDB opgevraagd worden en opgeslagen worden in de opnieuw online gekomen PDSDB. Een nadeel aan deze methode is wel dat wanneer de PDSDB langer dan drie uur offline is, er documenten in de DocumentDB opgeslagen zitten die niet in de PDSDB gestoken worden wanneer die terug online komt. Deze documenten zijn niet meet opvraagbaar. Het alternatief is dat wanneer de PDSDB terug online komt, de DocumentStorageManager vergelijkt welke documenten er voor de Registered Recipients in de PDSDB en de DocumentDB opgeslagen zijn. Het voordeel hierbij is dat alle documenten opvraagbaar blijven wanneer de PDSDB terug online komt. Het nadeel is dat dit meer werk inhoudt voor de DocumentStorageManager. Aangezien enkel drie uur vereist zijn, gaan we voor het eerste alternatief. De DocumentStorageManager ziet het schrijven in de PDSDB als impliciete ping-berichten waarbij het weet dat de PDSDB nog online is. Wanneer een schrijfrequest faalt, zal er een timer gestart wordt in de cache. Wanneer de PDSDB terug online is, zal deze een eenmalige heartbeat naar de DocumentStorageManager gestuurd worden, waarna hij weet dat hij terug kan schrijven. Hierna worden de documenten met verwijzingen in de cache geschreven naar de PDSDB door de DocumentStorageManager (die leest uit de DocumentDB en schrijft naar

de PDSDB). Merk op: als alternatief hadden we gedacht aan GEEN actieve heartbeat vanuit de PDSDB bij het online komen, maar bij een schrijfpoging naar de PDSDB als impliciete ping wanneer er een document opgeslagen wordt. Het probleem hierbij is dat wanneer er al even geen documenten genereerd zijn maar er toch noch documentreferenties in de cache zitten en op dat moment de PDSDB online komt, een registered recipient die documenten niet kan opvragen.

USE CASE 12: residual drivers: het opslaan van de recipient id's in de pdsdb –¿ hoort bij delivery-functionality. Ook het inloggen (authenticatie). USE CASE 13: residual drivers: we veronderstellen dat de DocumentStorageManager metadata bij elk document opslaat, like the name of the sender, the data range in which the document should be received and the document type –¿ hoort bij deliveryfunctionality. The meta-data is saved in both the pdsdb en database components, because when a user registers, this metadata has to be copied form the one to the other. Dit is is om gemakkelijk de documenten in de pdsdb te kunnen zoeken. USE CASE 14: residual driver: mark as received, puntje 3. –¿ beter te doen bij de deliveryfunctionality USE CASE 15: residual driver: tracking.

Alternatives considered

Alternatives for solution A discussion of the alternative solutions and why that were not selected.

3.2.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting decomposition.

ModuleB Per introduced component a paragraph describing its responsibilities.

ModuleC Per introduced component a paragraph describing its responsibilities.

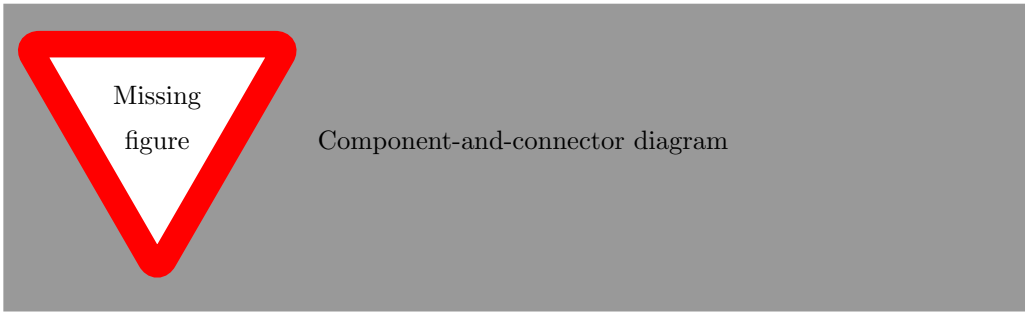


Figure 4: Component-and-connector diagram of this decomposition.

Behaviour If needed and explanation of the behaviour of certain aspects of the design so far.

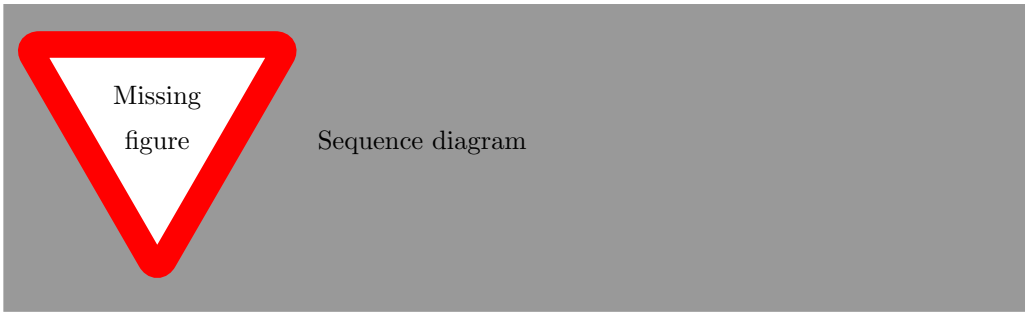


Figure 5: Sequence diagram illustrating a key behavioural aspect.

Deployment Rationale of the allocation of components to physical nodes.

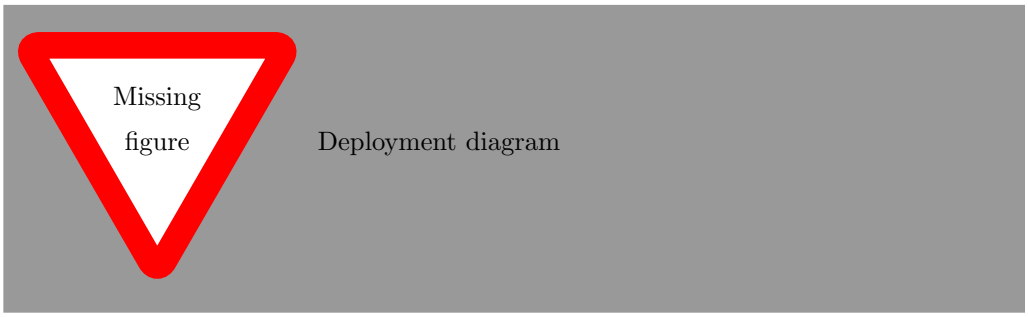


Figure 6: Deployment diagram of this decomposition.

3.2.5 Interfaces for child modules

ModuleB

- InterfaceA
  - `returnType operation1(ParamType param1)` throws TypeOfException
    - Effect: Describe the effect of calling this operation.
    - Exceptions:
      - TypeOfException: Describe when this exception is thrown.
  - `returnType operation2()`
    - Effect: Describe the effect of calling this operation.
    - Exceptions: None

3.2.6 Data type definitions

Describe per complex data type used in the interfaces what it represents.

returnType This data element represents X.

ParamType This data element represents Y.

3.2.7 Verify and refine

This section describes per component which (parts of) the remaining requirements it is responsible for.

ModuleB

- *Z1*: name
- *UCd*: name

ModuleC

- *UCba*: name
  Description which part of the original use case is the responsibility of this component.

3.3 Decomposition 3: ModuleA (X1, Y3, UCa, UCb, UCc)

3.3.1 Module to decompose

In this run we decompose `ModuleA`.

3.3.2 Selected architectural drivers

The non-functional drivers for this decomposition are:

- *X1*: name
- *Y3*: name

The related functional drivers are:

- *UCa*: name
- *UCb*: name
- *UCc*: name

Rationale A short discussion of why these drivers were selected for this decomposition.

3.3.3 Architectural design

Topic Discussion of the solution selected for (a part of) one of the architectural drivers.

Alternatives considered

Alternatives for solution A discussion of the alternative solutions and why that were not selected.

3.3.4 Instantiation and allocation of functionality

Decomposition Main aspects of the resulting decomposition.

ModuleB Per introduced component a paragraph describing its responsibilities.

ModuleC Per introduced component a paragraph describing its responsibilities.

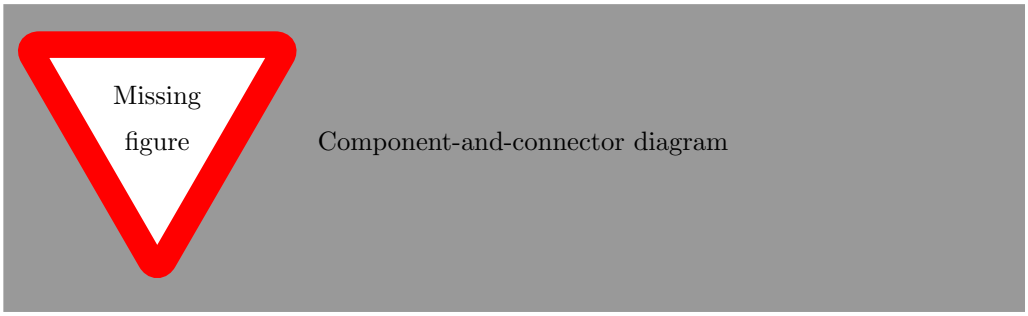


Figure 7: Component-and-connector diagram of this decomposition.

Behaviour If needed and explanation of the behaviour of certain aspects of the design so far.

Deployment Rationale of the allocation of components to physical nodes.

3.3.5 Interfaces for child modules

ModuleB

- InterfaceA
  - `returnType operation1(ParamType param1)` throws TypeOfException
    - Effect: Describe the effect of calling this operation.
    - Exceptions:

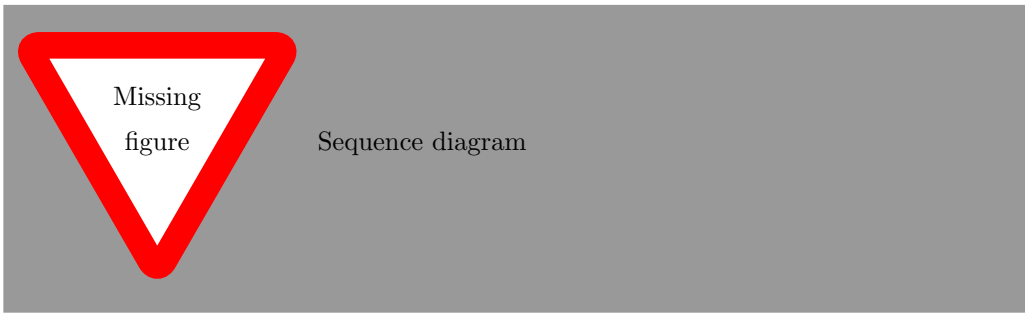


Figure 8: Sequence diagram illustrating a key behavioural aspect.

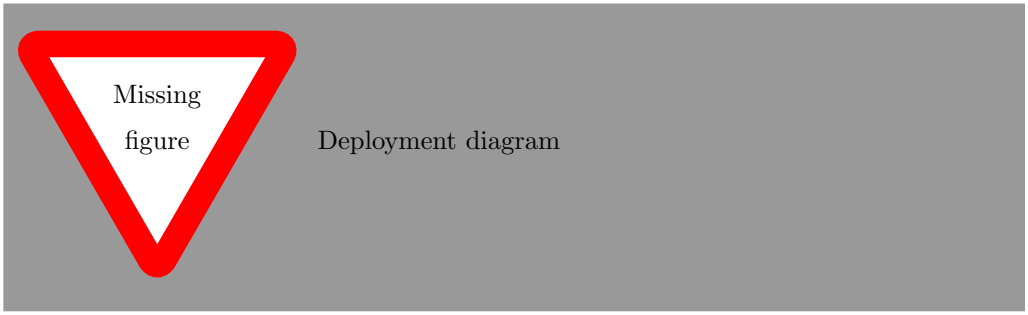


Figure 9: Deployment diagram of this decomposition.

        - TypeOfException: Describe when this exception is thrown.
    - `returnType operation2()`
        - Effect: Describe the effect of calling this operation.
        - Exceptions: None

3.3.6 Data type definitions

Describe per complex data type used in the interfaces what it represents.

returnType This data element represents X.

ParamType This data element represents Y.

3.3.7 Verify and refine

This section describes per component which (parts of) the remaining requirements it is responsible for.

ModuleB

- *Z1*: name
- *UCd*: name

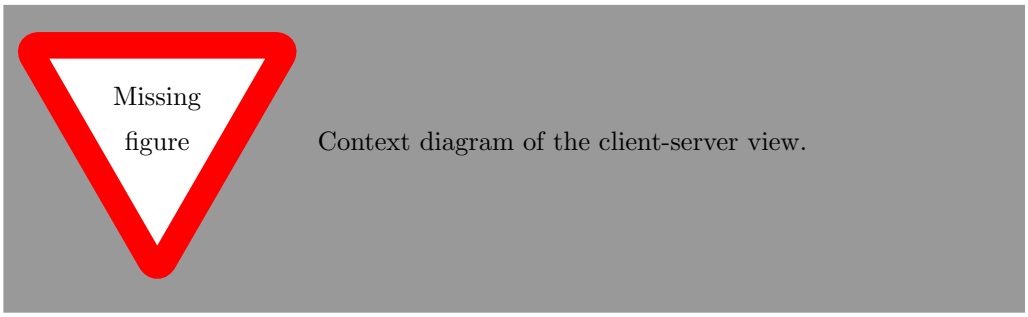
ModuleC

- *UCba*: name
  Description which part of the original use case is the responsibility of this component.

4 Client-server view (UML Component diagram)

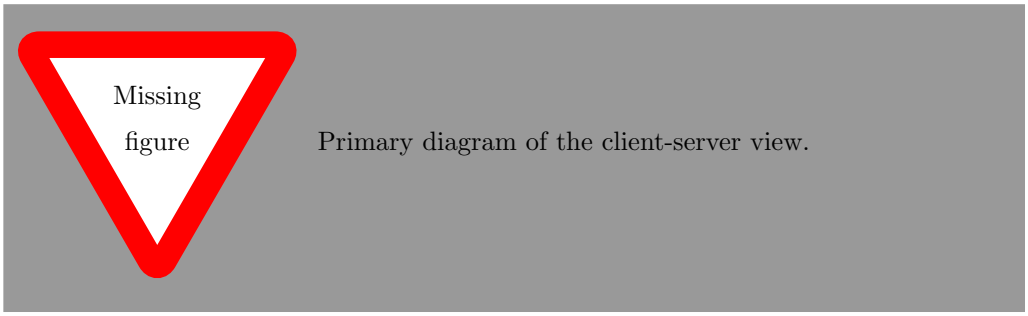
The context diagram of the client-server view. Discuss which components communicate with external components and what these external components represent.

The primary diagram and accompanying explanation.



Context diagram of the client-server view.

Figure 10: Context diagram for the client-server view.



Primary diagram of the client-server view.

Figure 11: Primary diagram of the client-server view.

4.1 Main architectural decisions

Discuss your architectural decisions for the most important requirements in more detail using the components of the client-server view. Pay attention to the solutions that you employed and the alternatives that you considered. The explanation here must be self-contained and complete. Imagine you had to describe how the architecture supports the core functionality to someone that is looking at the client-server view only. Hide unnecessary details (these should be shown in the decomposition view).

4.1.1 ReqX: requirement name

Describe the design choices related to *ReqX* together with the rationale of why these choices where made.

Alternatives considered

Alternative(s) for choice 1 Explain what alternative(s) you considered for this design choice and why they where not selected.

5 Decomposition view (UML Component diagram)

Discuss the decompositions of the components of the client-server view which you have further decomposed.

5.1 ComponentX

Describe the decomposition of `ComponentX` and how this relates to the requirements.

6 Deployment view (UML Deployment diagram)

Describe the context diagram for the deployment view. For example, which protocols are used for communication with external systems and why?

The primary deployment diagram itself and accompanying explanation. Pay attention to the parts of the deployment diagram which are crucial for achieving certain non-functional requirements. Also discuss any alternative deployments that you considered.

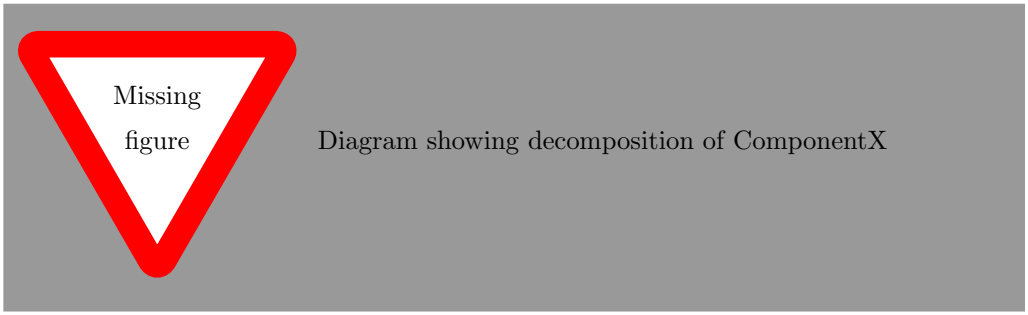
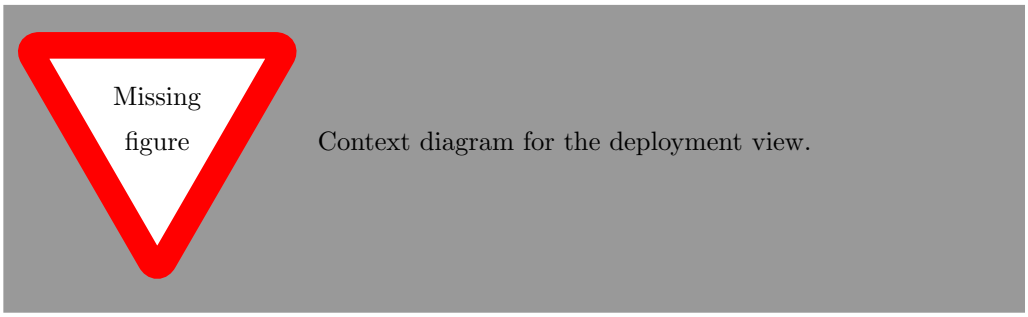


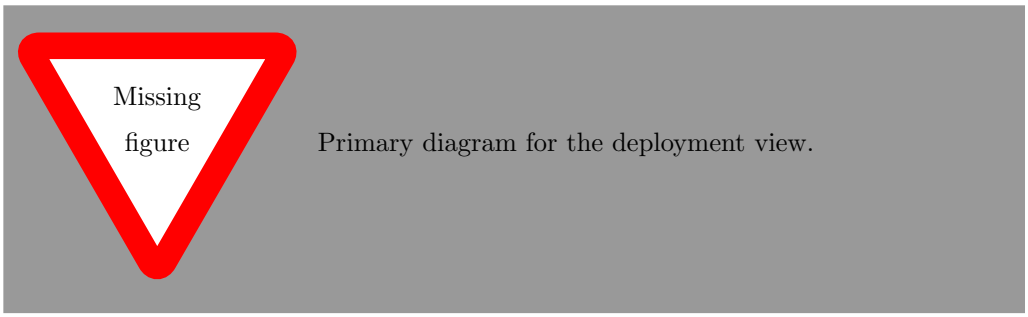
Diagram showing decomposition of ComponentX

Figure 12: Decomposition of `ComponentX`



Context diagram for the deployment view.

Figure 13: Context diagram for the deployment view.



Primary diagram for the deployment view.

Figure 14: Primary diagram for the deployment view.

7 Scenarios

Illustrate how your architecture fulfills the most important data flows. As a rule of thumb, focus on the scenario of the domain description. Describe the scenario in terms of architectural components using UML Sequence diagrams and further explain the most important interactions in text. Illustrating the scenarios serves as a quick validation of the completeness of your architecture. If you notice at this point that for some reason, certain functionality or qualities are not addressed sufficiently in your architecture, it suffices to document this, together with a rationale of why this is the case according to you. You do not have to further refine you architecture at this point.

7.1 Scenario 1

Shortly describe the scenario shown in this subsection. Show the complete scenario using one or more sequence diagrams.

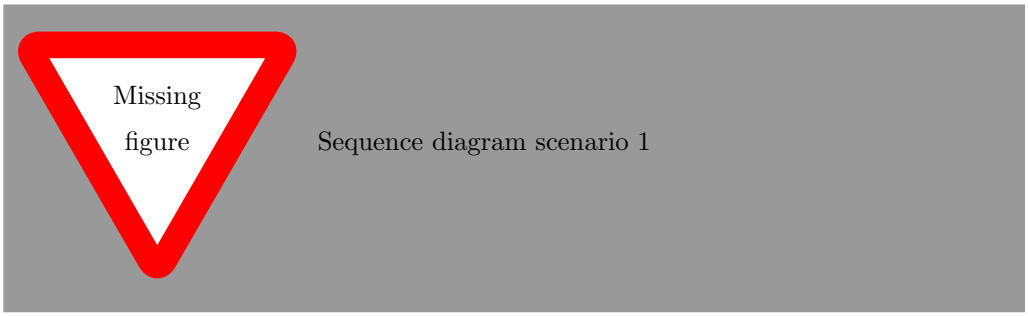


Figure 15: The system behavior for the first scenario.

A Element catalog

List all components and describe their responsibilities and provided interfaces. Per interface, list all methods using a Java-like syntax and describe their effect and exceptions if any. List all elements and interfaces alphabetically for ease of navigation.

A.1 Completer

- **Description:** Responsibilities of the component.
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

Provided interfaces

- Complete
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
    - `void operation2(ParamType2 param)`
      - Effect: Describe the effect of the operation
      - Exceptions: None

A.2 DocumentGenerationManager

- **Description:** The `DocumentGenerationManager` monitors the availability of the `Generator` using ping/echo. The `DocumentGenerationManager` keeps track of the jobs assigned to and being processed by the `Generators`. The `DocumentGenerationManager` assigns jobs to the `Generators` in groups of more than one job that are part of the same batch.
- **Super-component:** None
- **Sub-components:** `Completer`, `GenerationManager`, `KeyCache`, `Scheduler`, `TemplateCache`

Provided interfaces

- InsertJobs
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
    - `void operation2(ParamType2 param)`
      - Effect: Describe the effect of the operation
      - Exceptions: None
- NotifyCompleted
  - `JobBatch notifyCompletedAndGiveMeMore(GeneratorId id)`
    - Effect: Describe the effect of the operation
    - Exceptions: None

A.3 Generator

- **Description:** Generates the documents and forwards them to `OtherFunctionality` to store and deliver them.
- **Super-component:** None
- **Sub-components:** None

Provided interfaces

- AssignJobs
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
    - `void operation2(ParamType2 param)`
      - Effect: Describe the effect of the operation
      - Exceptions: None
- Startup/ShutDown
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- Ping
  - `Echo ping()`
    - Effect: The `Generator` will respond to the ping request by sending an echo response. This is used by the `GeneratorManager` to check whether the `Generator` is available.
    - Exceptions: None

A.4 GenerationManager

- **Description:** Responsibilities of the component.
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

Provided interfaces

- AssignJobs
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
    - `void operation2(ParamType2 param)`
      - Effect: Describe the effect of the operation
      - Exceptions: None
- Startup/ShutDown
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- Ping
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

A.5 KeyCache

- **Description:** Responsibilities of the component.
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

Provided interfaces

- GetKey
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
    - `void operation2(ParamType2 param)`
      - Effect: Describe the effect of the operation
      - Exceptions: None

A.6 PDSDB

- **Description:** Responsibilities of the component.
- **Super-component:** None
- **Sub-components:** `PDSDBReplica`, `PDSLongTermDocumentManager`, `PDSReplicationManager`

Provided interfaces

- DocumentMgmt
  - `void storeDocument(DocumentId id, Document doc, MetaData md)`
    - Effect: The `PDSDB` will store the given document`doc` together with the provided metadata `md`.
    - Exceptions: None
  - `Tuple<Document, MetaData> getDocument(DocumentId id)`
    - Effect: Describe the effect of calling this operation.
    - Exceptions: None
  - `void markReceived(DocumentId id)`
    - Effect: Describe the effect of calling this operation.
    - Exceptions: None

A.7 PDSDBReplica

- **Description:** Responsibilities of the component.
- **Super-component:** `PDSDB`
- **Sub-components:** None

Provided interfaces

- ExtendedDocumentMgmt
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
    - `void operation2(ParamType2 param)`
      - Effect: Describe the effect of the operation
      - Exceptions: None
- Ping
  - `Echo ping()`
    - Effect: The `PDSDBReplica` will respond to the ping request by sending an echo response. This is used by the `PDSReplicationManager` to check whether the `PDSDBReplica` is available.
    - Exceptions: None

A.8 PDSLongTermDocumentManager

- **Description:** Responsibilities of the component.
- **Super-component:** `PDSDB`
- **Sub-components:** None

Provided interfaces

- DocumentMgmt
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
    - `void operation2(ParamType2 param)`
      - Effect: Describe the effect of the operation
      - Exceptions: None

A.9 PDSReplicationManager

- **Description:** Responsibilities of the component.
- **Super-component:** `PDSDB`
- **Sub-components:** None

Provided interfaces

- ExtendedDocumentMgmt
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
    - `void operation2(ParamType2 param)`
      - Effect: Describe the effect of the operation
      - Exceptions: None

A.10 OtherFunctionality

- **Description:** Responsibilities of the component.
- **Super-component:** None
- **Sub-components:** the direct sub-components, if any.

Provided interfaces

- FinalizeDocument
  - `void storeAndDeliverDocument(JonId jobid, Document doc) throws SomeException`
    - Effect: The `OtherFunctionality` will store the given document `document` and deliver it.
    - Exceptions: None
    - `void generationError(JobId jobid, Error error)`
      - Effect: Describe the effect of the operation
      - Exceptions: None
- GetBatchData
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- GetTemplate
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- GetKey
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- setStatus
  - `returntType2 operation3()`

        * Effect: Describe the effect of the operation
        * Exceptions: None

- NotifyOperator
    - `returntType2 operation3()`
        * Effect: Describe the effect of the operation
        * Exceptions: None

A.11 Scheduler

- **Description:** Responsibilities of the component.
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

Provided interfaces

- GetNextJobs
    - `returntType1 operation1(ParamType param) throws SomeException`
        * Effect: Describe the effect of the operation
        * Exceptions:
            · SomeException: Describe when the exception is thrown.
        * `void operation2(ParamType2 param)`
            · Effect: Describe the effect of the operation
            · Exceptions: None
- InsertJobs
    - `returntType2 operation3()`
        * Effect: Describe the effect of the operation
        * Exceptions: None
- GetStatistics
    - `returntType2 operation3()`
        * Effect: Describe the effect of the operation
        * Exceptions: None

A.12 TemplateCache

- **Description:** Responsibilities of the component.
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

Provided interfaces

- GetTemplate
    - `returntType1 operation1(ParamType param) throws SomeException`
        * Effect: Describe the effect of the operation
        * Exceptions:
            · SomeException: Describe when the exception is thrown.
        * `void operation2(ParamType2 param)`
            · Effect: Describe the effect of the operation
            · Exceptions: None

B Defined data types

List and describe all data types defined in your interface specifications. List them alphabetically for ease of navigation.

- `BatchId`: Description of data type.
- `Document`: Description of data type.
- `Echo`: The response to a ping message. This data element does not contain any meaningful data.
- `Error`: Description of data type.
- `GeneratordId`: Description of data type.
- `JobBatch`: Description of data type.
- `JobId`: Description of data type.
- `MetaData`: Description of data type.