KU LEUVEN

DEPARTMENT OF COMPUTER SCIENCE

Patient Monitoring Service (PMS)

Part 2

Architecture evaluation

GHARANFOLI–NGO–NICOLAS

ACADEMIC YEAR 2021 – 2022

H09B5B: SOFTWARE ARCHITECTUUR
H07Z9B: SOFTWARE ARCHITECTURE

Sobhan GHARANFOLI (r0734169)
Triet NGO (r0869104)
Georgio NICOLAS (r0875445)

Analysis and Evaluation

Q1. What is the distinction between ClinicalModels and MLModels (if any)?

- **Answer:** A MLModel is a machine learning model that is used to make predictions while a ClinicalModel is a set of parameters that can be used for a set of MLModels which are contained inside a ClinicalModel.
- **Evidence and assumptions:** E.6 Data types: ClinicalModel and MLModel
- **Opinion and improvements:**

Q2. Where are the ClinicalModels stored? Who creates clinical models in the system? Will a single ClinicalModel be used for different patients?

- **Answer:** Component responsible for ClinicalModel storage: **ClinicalModelDB**, and this component is deployed on node **RiskEstimationManagmentNode** (**Dev Deployment**)

  ClinicalModelDB provides interface **ClinicalModelStorage** and this interface is used only by **ClinicalModelCache**. The operation that this interface provides is **getAssignedClinicalModelsAndConfigForPatient** (a get operation to retrieve models for a patient). Since there is no create operation, there is no component in the current design that can create a clinical models.

  Based on the deployment diagram there are 20 patients and 3 ClinicalModels. We can assume that there will exist a model that will be used for more than 1 person.

- **Evidence and assumptions:** Deployment diagram C.3, C.4

Q3. The system uses machine learning based models to perform risk assessment. Are different types of models (non-ML) also supported and if not, would adding support be impactful? Where are these supported MLModels stored?

- **Answer:** The system currently supports ML models only, which are referenced by ID strings that are stored in **applicableMLModelIds** under **ClinicalModel**. Adding support to different kinds of models could be implemented in different ways. One way to do that is by modeling a Model superclass and referencing specialized instances of it (Generic, ML, Statistical, etc...) by ID in the ClinicalModel. This would require changes in the functions which fetch models, start jobs, and process data with MLModels, to allow them to support the suggested Model superclass and behave differently based on the type of each instance of a model. Another way to achieve this goal is to "morph" other types of models into MLModels("The actual internal encoding of the model itself is left open"). This workaround would require less modification to the architecture but will definitely include some inconveniences, as the users would be unable to make use of any exclusive functionality provided by a different type of model.

  MLModels are currently stored in the **MLModelRepository**.

- **Evidence and assumptions: E.6 Datatypes**

Q4. When they are needed for a risk estimation job, how does the RiskEstima- tionProcessor get the applicable models (both ClinicalModels and MLModels)? Is it fetching or loading these for every job that is executed, and why (not)?

- **Answer:** When a component calls **launchRiskEstimation**, **ClinicalJobCreator** will get the models and configs from **ClinicalModelCache**, if it is not cache, **ClinicalModelCache** will retrieve it from **ClinicalModelDB** and create a Job. When a job is called **JobProcessor** will call **JobMLModelSelector** and **MLModelRetrieval** to select **MLModel**s from **MLModelMgmt**.

  ClinicalModels are stored in **ClinicalModelDB** and cached by **ClinicalModelCache** and only retrieved when it is uncache so not all jobs will require retrieval.

  MLModels are retrieved every time from the local storage and only getting update on call by operation **synchronizeModels**.

- **Evidence and assumptions: sequence diagram D.1, E.1.2**

Q5. Could the clinical models be executed on the patient gateway (smartphone)? Which architectural considerations play a role in this decision? Can 'parts of clin- ical models' be performed on the gateway (i.e., could the required computational load be distributed over the involved devices in some way)?

- **Answer:** While it is not possible for the patient gateway to execute the clinical model with the current design. Theoretically, it is possible so add modifications so that the patient gateway can execute clinical models but there are disadvantages regarding Security, Resources (Memory, Storage, Processing power), Reliability

  - MLModels need to be transferred to the client gateway (MLModels can take up lots of storage, and could be leaked/manipulated)
  - Client gateway needs to have enough computing power
  - Make sure all client gateways have up to date data and models
  - Client gateway needs to store all the data from sensors to serve as input to the MLModels.
  - System would be much more prone to false alerts and undetected emergencies.

  For **ClinicalJob**s that require aggregate data (e.g. mean heart rate over a day, maximum blood pressure over a week, etc.), we can utilize the client gateway to store, compute, and transfer this data precomputed to the server when required. (If the ClinicalModelJob is modified to support aggregate data)

Q6. Does the developer need to provide the functionality for executing or applying the applicable models (both ClinicalModels and MLModels) from scratch? If not, who develops this?

- **Answer: MLModel**s are executed through the **MLAPI** which is provided and implemented by **ML-Provider**.

  ClinicalModels are applied/executed under the **RiskEstimationProcessor** which is provided and implemented by PMS.

- **Evidence and assumptions:E.1.10, E.2.4, E.3.13**

Q7. Which non-functional considerations have been taken into account in the initial design of the RiskEstimatorProcessor? Per discussed non-functional aspect, indicate which specific view and diagram you used to find out.

- **Answer:** From the deployment diagram C.2. Multiple instances of **RiskEstimatorProcesso**r are deployed in the **RiskEstimationProcessorNode**. This consider the following non-functional aspect:

  - Availability: If one of the **RiskEstimatorProcesso**r instances crashes, the others can take over the job.
  - Performance: We can run multiple queries with in different nodes **RiskEstimatorProcessor** which increase the processing time when multiple clients request prediction at the same time.
  - Reliability: We can use Bagging which employs multiple **RiskEstimatorProcessor** nodes to fetch ML predictions based on the data while using different parameters and aggregating the results to increase the accuracy and confidence of the ML-based decisions.

- **Evidence and assumptions: Deployment diagram C.2**

Q8. Would it be possible to change to a different MLaaS? What is the architec- tural impact of such a change?

- **Answer:** Yes, the architectural change would revolve From diagram B.2. The modules that requires interface **MLAPI**, **MLaaSService**, **MLaaSAPI** from the modules **MLaaSLibrary**, **MLLibrary** that are provided by the **MLaaS**. When we change the MLaaS, these components and interfaces will be different. Hence the component **MLModelUpdateProcessor**, **ModelUpdater** which require these interfaces will have to implement different interfaces from the new provider. Additionally, switching the MLaaS provider might require us to make some changes relative to storing and caching **MLModel**s locally which imply changes to the **MLCodeMgmt** interface required by the **MLModelStorageManager**.

- **Evidence and assumptions: diagram B.2, E 2.8, E.3.11, E.3.12**

Q9. Concrete algorithms have been abstracted. Is the selection of ML algorithms architecturally significant? If so, which non-functional aspects play a role?

- **Answer: JobMLModelSelector** is responsible for selecting the appropriate ML algorithms for predictions. The ML provider will implement the algorithms and provide us with a black box solution which we can interface with through an API, in addition to certain guarantees concerning the quality of their service. If the algorithm would satisfy or exceed our reliability, performance and interoperability requirements, then there is no direct impact for using one ML algorithm over another, since we only need to adhere to their provided API.
- **Evidence and assumptions: E.2.1**

Q10. What is the position of the MLaasLibrary in the context diagram of the client-server view, and in the context diagram of the decomposition view, and explain why. What does this model element represent?

- **Answer:** Component MLModelManager includes module MLModelUpdateProcessor which uses the interface MLaaSAPI that is provided by MLaaSLibrary
  - MLModelManager included in the client-server view.
  - Decompose MLModelManager we get MLModelUpdateProcessor communicate with the external MLaaSAPI which is provided by MLaaSLibrary.
- **Evidence and assumptions**: the context diagram of the client-server view, and in the context diagram of the decomposition view.

Q11. What is the position of the MLLibrary in the context diagram of the client-server view, and in the context diagram of the decomposition view, and explain why. What does this model element represent?

- **Answer:** Component **MLModelManager** includes module **SensorDataRiskAssessment** which uses the interface **MLAPI** that is provided by **MLLibrary**
  - **MLModelManager** included in the client-server view.
  - Decompose **MLModelManager** we get **MLModelUpdateProcessor** communicate with the external **MLaaSAPI** which is provided by **MLaaSLibrary**.
- **Evidence and assumptions**: the context diagram of the client-server view, and in the context diagram of the decomposition view.

Q12. What is the position of the MLaaS and its node in the context diagram of the deployment view, and explain why. What do these model element represent?

- **Answer:**
  - In the context diagram of the deployment view, the **MLaaS** is a component that is inside the **MLaaSCloud** node. The **MLaaSCloud** receives requests from our PMS via the **MLaaSConnectorNode** node which deploys the process **MLModelManager**.
  - In other words, the **MLaaS** component is a run-time process that is deployed under the **MLaaSCloud** node and used by PMS.
- **Evidence and assumptions**:the context diagram of the deployment view.

Q13. How does the system ensure an adequate throughput of risk calculation jobs?

- **Answer:** In the Primary deployment diagram, we can see in the **RiskEstimationProcessorNode** there are multiple deployment of the component **RiskEstimationProcessor**. This multiple instances will get jobs that are schedule by the processes in the node **RiskEstimationMgmtNode**. Therefore, in case of differences amount incoming requests, the scheduler will distribute jobs throughout these **RiskEstimationProcessorNode** to ensure an an adequate throughput of risk calculation jobs.

- **Evidence and assumptions: E.4, E.1**

Q14. How does the system react if there are too many processing jobs?

- **Answer:** In case of too many process jobs, the **RiskEstimationScheduler** that is responsible for taking the next job from the queue for estimation. The **RiskEstimationScheduler** also keep track of the throughput of incoming jobs and change the status between FIFO and overload modus. In overload modus, the **RiskEstimationScheduler** will use the following dynamic: earliest deadline first and enqueues jobs of patient with a high risk level with an earlier deadline (2 instead of 5 minutes) to prioritize them over patients with lower risk levels
- **Evidence and assumptions: E.1.11**

Q15. What if the system resources (CPU, memory, storage) for performing the risk calculations prove to be insufficient?

- **Answer:** The provided architecture does not provide information that covers the situation when the system does not have enough resources to perform risk calculations (which run on **RiskEstimationProcessorNode** instances). It's safe to assume, based on the primary deployment diagram, that multiple **RiskEstimationProcessorNode** instances may be deployed. However if a certain input for **RiskEstimationProcessorNode** causes it to starve, running another instance with the same resources will also starve. Since the architecture does not account for this sort of problem, we might reach a looping state where a **RiskEstimationProcessorNode** is launched, gets starved, craches, then another one with the same input is launched...
- **Evidence and assumptions: E.4.10**

Q16. What happens when risk calculation jobs fail (e.g., due to crashes or bugs)? Does it affect other jobs?

- **Answer:** In Primary deployment diagram, the **RiskEstimationMgmtNode** connects to multiple **RiskEstimationProcessorNode** nodes. The **RiskEstimationMgmtNode** will distribute the jobs to the **RiskEstimationProcessorNode** nodes based on the throughput.

  Therefore, in case of a job crashes or bugs, the **RiskEstimationProcessorNode** that is responsible for processing the "failed" job will fail. Since the **RiskEstimationProcessorNode** nodes are separated from each other, the failure on one node won't cause failure on other nodes. The Scheduler in the node **RiskEstimationMgmtNode** can dispatch the fail job again in other another **RiskEstimationProcessorNode** for retrying.
- **Evidence and assumptions:Primary deployment diagram, E.4**

Q17. The system uses machine learning based models to perform risk assessment. Are different types of models (non-ML) also supported and if not, would adding support be impactful?

- **Answer:**
  - Based on the current architecture, we only use ML based models for risk assessment and there are no different types of models (non-ML) that are supported by PMS.
  - Adding other models will cause changes in the component **RiskEstimationProcessor**. At the moment, the modules in **RiskEstimationProcessor** are designed to fit the ML process:
    * **JobMLModelSelector** for selecting ML model.
    * **JobProcessor** for coordinates the execution of **ClinicalModel**s.
    * **MLModelRetrieval** for to retrieve and update ML models. **SensorDataRiskAssessment** for applying the ML model to produce the result.
  - Based on this structure, we will need to change **JobProcessor** to coordinate and ensure ClincalModels also apply for new models (non-ML). A new **JobModelSelector** to select between ML or non-ML. Adjust **SensorDataRiskAssessment** to apply new models.
  - In conclusion, the adjustment will drastically change the **RiskEstimationProcessor** component: both design and functionalities of its modules.

- **Evidence and assumptions: E.1.10, E.2.1, E.2.2, E.2.6, E.2.10**

Q18. Do changes made to clinical models affect ongoing calculations? Why (not)? When will changes to clinical models affect new risk estimation jobs?

- **Answer:** In the D.1. sequence diagram, when creating a **ClinicalJobs**, the **ClinicalJobCreator** fetch the **ClinicalModel** from either the **ClinicalModelCache** or **ClinicalModelDB**. After fetching, changes made to the ClinicalModel in the database won't affect the created or ongoing jobs since the configurations and parameters for the ClinicalModel are fixed. While the jobs depending on the previous version of the clinical model are running, and the cache is full, new jobs depending on the new clinical models will have to fetch them directly from the database. When no more jobs running depend on the old clinical models, the cache can be invalidated by calling **invalidateCacheEntries()** to clear the cache and fetch the new models. Essentially, until the cache is invalidated, the changes in ClinicalModel won't affect the performance of current jobs.

- **Evidence and assumptions: D.1. sequence diagram**

- **Opinion and improvements**: we assumed that **invalidateCacheEntries()** would be called by some components but it did not based on the current architecture. One suggestion would be to have a process fetch and compare the cache with the up-to-date model.

Q19. Which subsystems will be affected by modifications to the clinical risk models?

- **Answer:** In case of changes in values of the **ClinicalModel**, the previous question addressed this problem. From diagram Primary deployment diagram, changes in structure of ClinicalModel will affect the all the components that are deployed on node **RiskEstimationMgmtNode** and **RiskEstimationProcessorNode** since all of these components employ the same structure of a ClinicalModel from creating job to fetching, processing and producing prediction results.

- **Evidence and assumptions: Primary deployment diagram**

Q20. Where is the ClinicalModelCache deployed? Which other component(s) should be deployed together with the cache?

- **Answer:** It is deployed in the following instances: **RiskEstimationManagementNode** (Pilot Deployment), **RiskEstimationManagmentNode** (Dev Deployment), **RiskEstimationMgmtNode**. **ClinicalModelDB** needs to be deployed together with the cache because it contains the actual data (Clinical Models) that needs to be cached.

- **Evidence and assumptions: Pilot Deployment diagram, Dev Deployment diagram**

E1 Design Extension

Extend this initial design with a dedicated dashboard that allows pms Telemedicine operators to overview the ongoing risk estimation processes, monitor the performance of the subsystem in terms of relevant system metrics (e.g., average job execution time, system status). Make sure that this is a web front- end (e.g., based on web frameworks such as Kibana, Zabbix) has read-only access for inspecting the system with no side-effects (e.g., no configuration actions or changes to the operational system).

Design Decisions

For this extension, we decided to create 3 new components responsible for different aspect of the Telemedicine extionsion: a database for storing records **TelemedicineDB**, a Central Service for requests from PMS and Telemedicine clients **TelemedicineService**, a front-end service **TelemedicineFronendService**

The Central Service is splited in to 2 modules 3: **TelemedicineUpdate** which is responsible for update operators from PMS whereas **TelemedicineQuery** handles queries from front-end service (e.g. performance monitors, system metrics, ongoing jobs, etc.). The decision being we need a processor in each of the relevant areas to restrict the operators by providing different interfaces for example read-only access operations.

A database is provided to store objects type Status which consist of ClinicalModelJob, start and end time. We decided to separate this database from the PMS database for performance issues since this extension is served as monitoring services which should not affect the performance of the PMS services. Information are requested and processed separately in components.

Discussion

In this section, we will discuss the changes in diagram and architecture.

- The components of TelemedicinePMS is described in Firgure 1. TelemedicineService provides TelemedQueryAPI which contains read-only operators (for system metrics, ongoing processes, etc.) for TelemedicineFrontendService which used them to distribute the result to the clients. Furthermore, TelemedicineDB provides simple read/write API so that TelemedicineService can use.
- Now let's deconstruct TelemedicineService in Firgure 3. TelemedicineService consists of 2 modules: TelemedicineUpdate which responsible for providing operators for PMS (e.g. read write operators for monitoring), TelemedicineQuery provides read-only operators for front-end.
- To integrate Telemedicine to PMS systems, we integrate the TelemedicineUpdate module in the RiskEstimationProcessor which can be seen in Figure 4. We create a module JobStatusProcessor which communicate with Jobprocessor so that we can invoke TelemedPmsAPI when there is a job starts or ends.
- For deployment (Figure 5), we deployed all 3 components of Telemedicine Extension to TelemedicinePMS node. This can be improved by provide multiple component of TelemedicineService if the load of it is extreme but we also have to take into account the cost of multi-processor extension.
- Finally, we adjust the sequence diagram 6 so that we can monitor ongoing jobs. When we started a job, invoke startJob and provide the current ClinicalJob and the current timestamp so that we can save in Telemedicine that the job is running. At the end of the job when the the result is determined, we invoke endJob to inform Telemedicine that the job ended at the current timestamp.
- This example provide an overview of how on going jobs can be monitored without record too many parameters or interfering with PMS infrastructure. For metrics, we can compute based on the records, for example, throughput during 2 timestamp: we can query all records that start and end between the 2 timestamp, or any failed job: we can query all records that do not have an end time.

Final reflections

Q21. Which architectural tactics or patterns did you recognize? For which quality attributes do you think they were introduced?

- For **Detect Faults**, we are asked to design an extension for monitoring the PMS system and subsystems.
- For performance tactics, Increase resources such as deploy multiple RiskAssessmentProcessor components or Increase concurrency with a JobScheduler.
- For patterns, a simplest one would be the Client-Server which is used across the implementation of PMS.

Q22. How would you summarize the strengths and weaknesses of this architecture? Are there other aspects of the architecture that strike you as either genius or odd? If so, list these below and briefly explain.

- One of the strength of this architecture is the details of documentation (multiple UML diagrams and detail description) which most of the project would not have. The components are separate carefully while having separating between ongoing process and structure of a component.
- One weakness of the architecture is the rigid design. When working on the evaluation questions, it is very clear that the extension problem or changes assumption questions created problems across the design (one example would be using add a non-ML model).

- One genius of this architecture is the robustness of the components when deploy in different scenarios. From the deployment diagram provided to us, we can observe how different the deployment is to one another (the amount of Processor depends on number of patient, the distribution of nodes and components, etc.).

Which follow-up questions would you ask the software architect that has created this initial architecture (if any)? Which assumptions or interpretations made during the evaluation exercise would you like to double-check? As we discussed all the evaluation questions with the teaching assistants during lab sessions, we received lots of feedback and confirmation for our assumption. As a result we do not have any further questions.

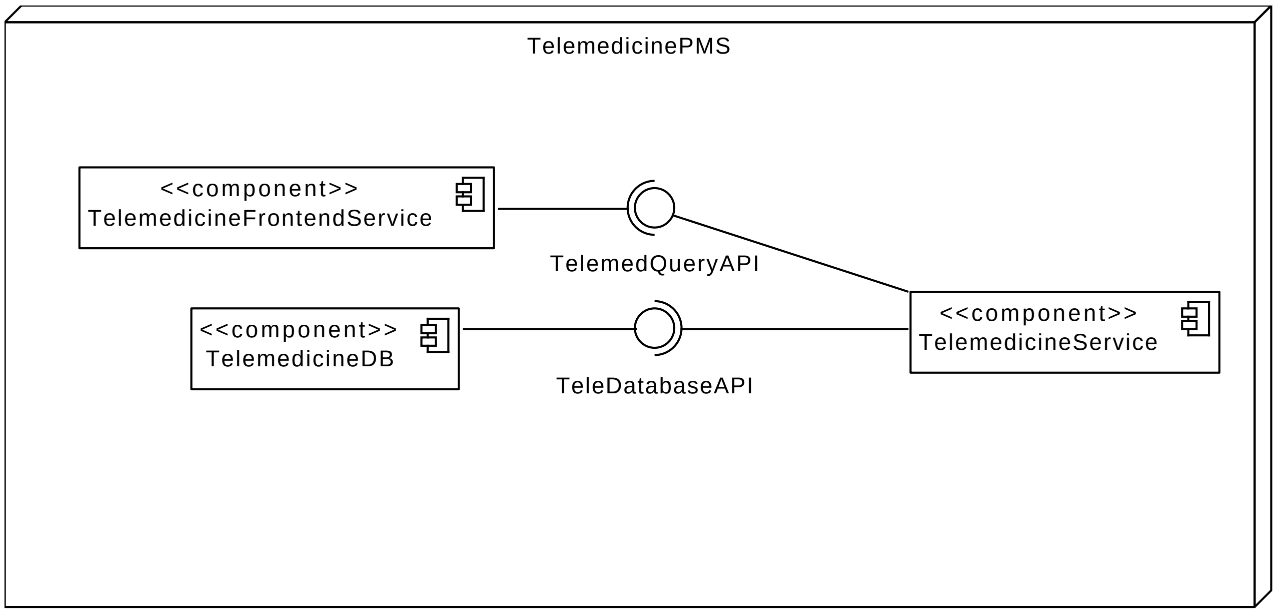

Figure 1: Component diagram of TelemedicinePMS

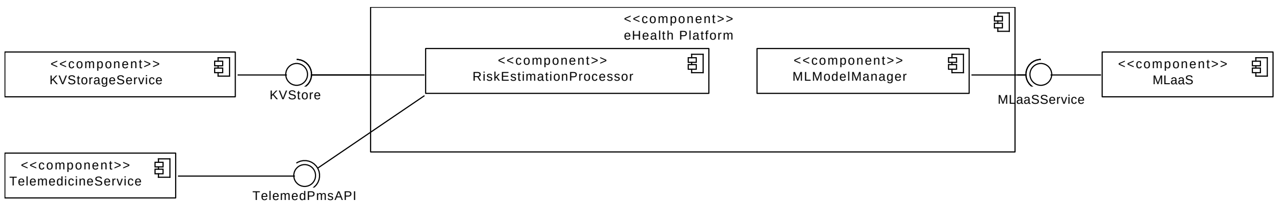

Figure 2: Updated Context diagram for the client-server view

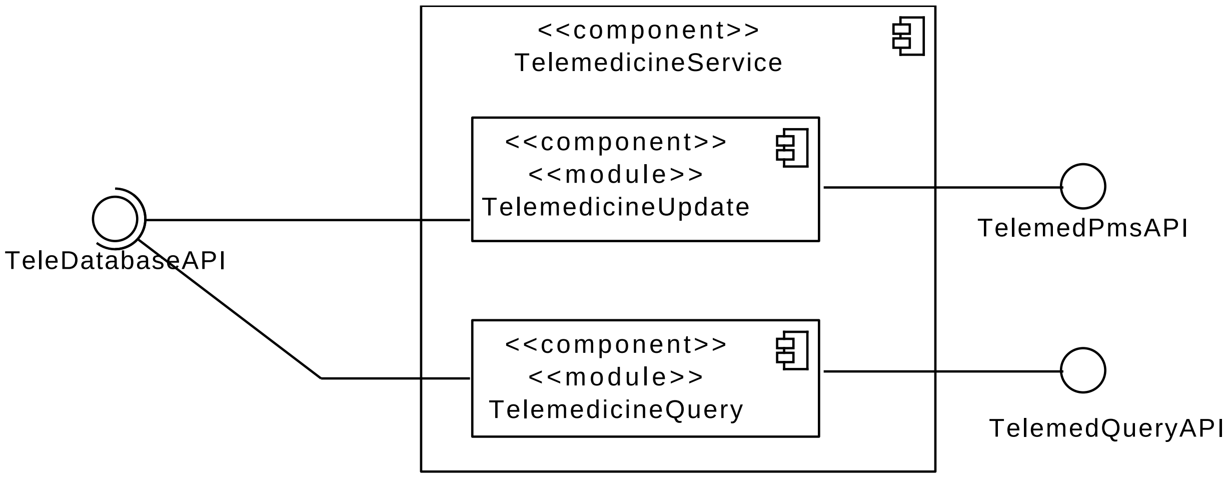

Figure 3: Decomposition of TelemedicineService

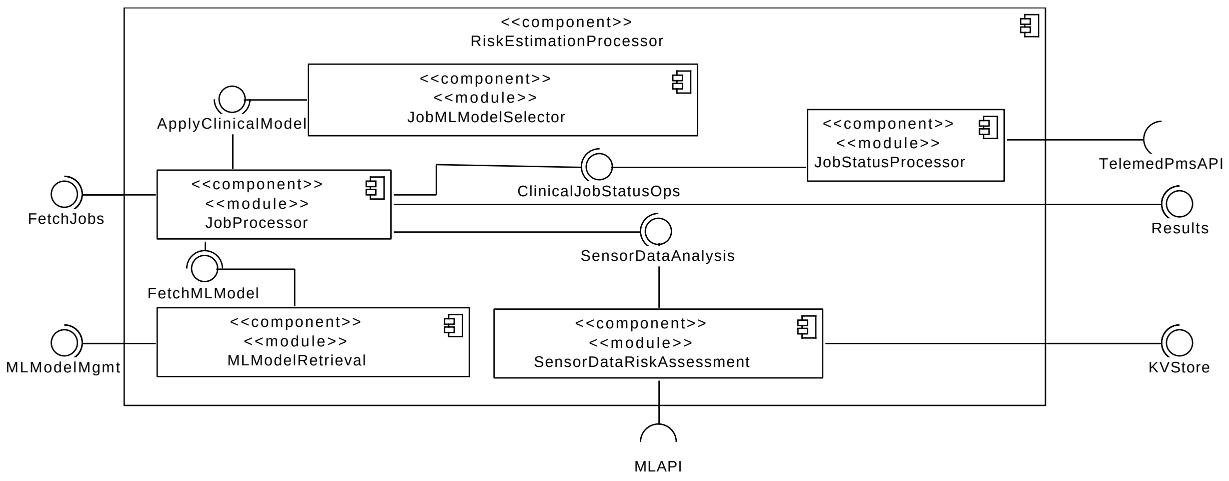

Figure 4: Decomposition of the RiskEstimationProcessor

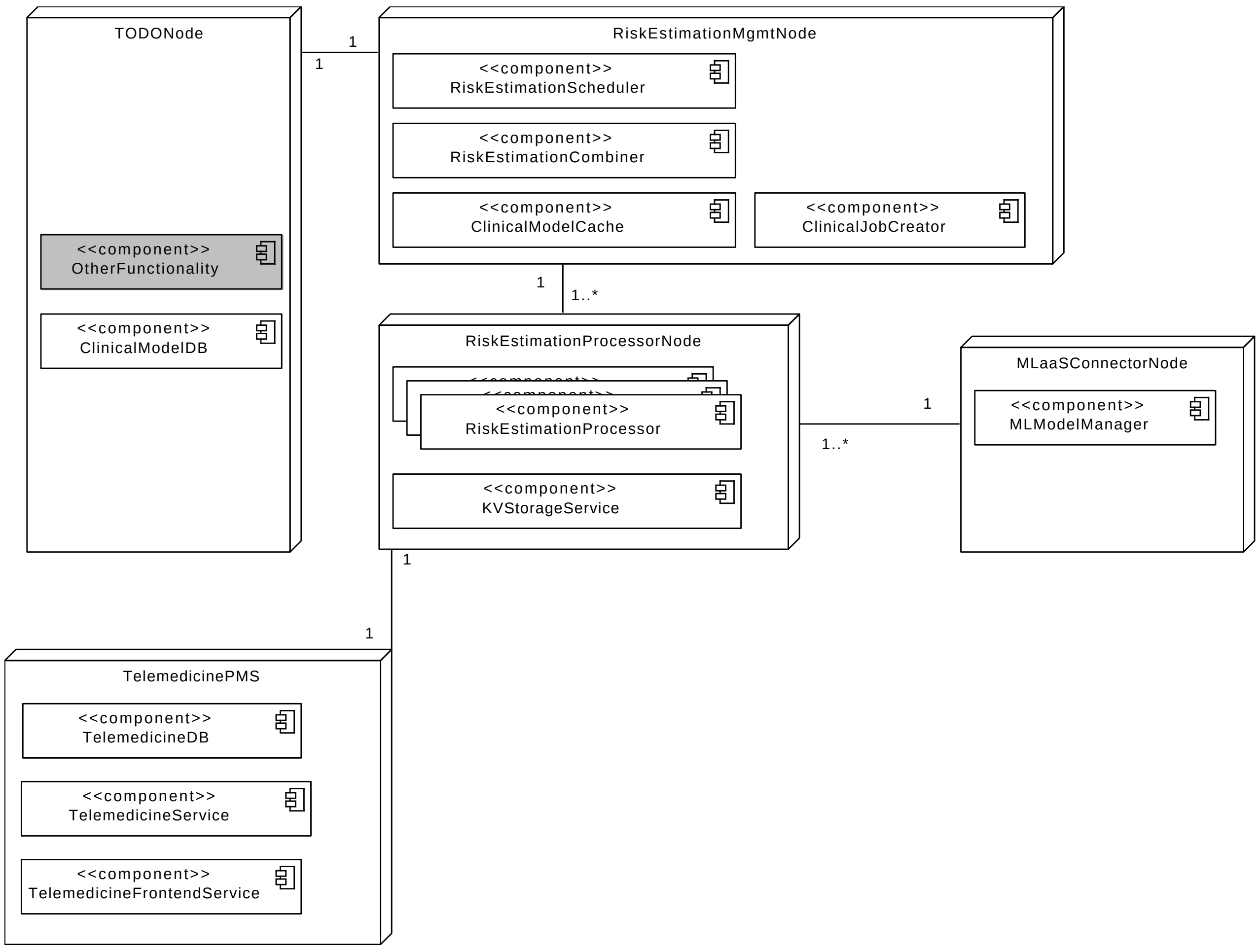

Figure 5: Updated Primary deployment diagram

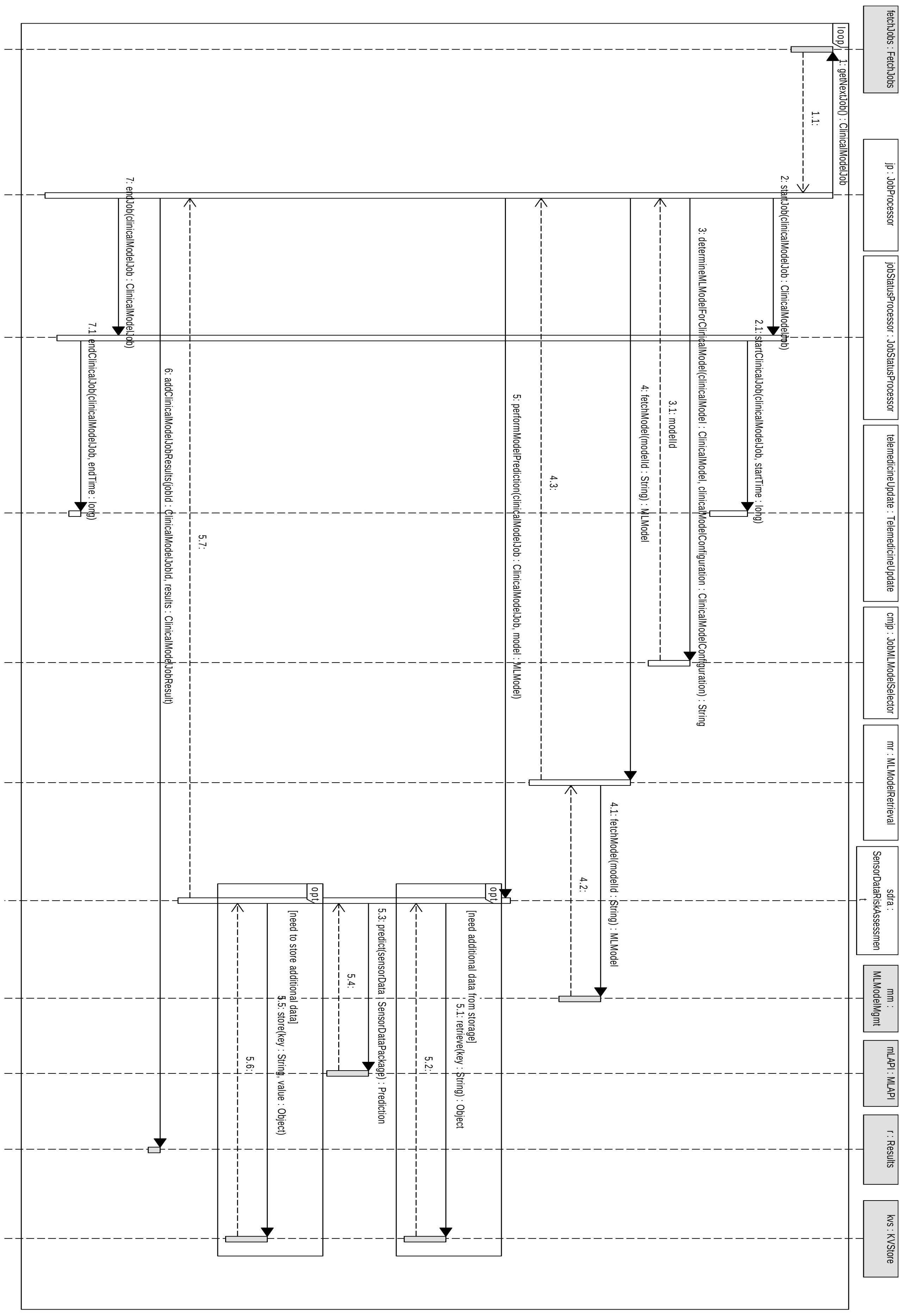
fetchJobs : FetchJobs
jp : JobProcessor
jobStatusProcessor : JobStatusProcessor
telemedicineUpdate : TelemedicineUpdate
cmjp : JobMLModelSelector
mr : MLModelRetrieval
sdra : SensorDataRiskAssessment
mm : MLModelMgmt
mLAPI : MLAPI
r : Results
kvs : KVStore
loop
1: getNextJob() : ClinicalModelJob
1.1:
2: startJob(clinicalModelJob : ClinicalModelJob)
2.1: startClinicalJob(clinicalModelJob, startTime : long)
3: determineMLModelForClinicalModel(clinicalModel : ClinicalModel, clinicalModelConfiguration : ClinicalModelConfiguration) : String
3.1: modelId
4: fetchModel(modelId : String) : MLModel
4.1: fetchModel(modelId : String) : MLModel
4.2:
4.3:
5: performModelPrediction(clinicalModelJob : ClinicalModelJob, model : MLModel)
opt
[need additional data from storage]
5.1: retrieve(key : String) : Object
5.2:
5.3: predict(sensorData : SensorDataPackage) : Prediction
5.4:
opt
[need to store additional data]
5.5: store(key : String, value : Object)
5.6:
5.7:
6: addClinicalModelJobResults(jobId : ClinicalModelJobId, results : ClinicalModelJobResult)
7: endJob(clinicalModelJob : ClinicalModelJob)
7.1: endClinicalJob(clinicalModelJob, endTime : long)

Figure 6: Sequence diagram compute clinical model result