# Dialog Management - Knowledge Integration Interactions

The interaction between DM and KI[[1]](#footnote-1) (Figure 1) mainly aims to (i) feed the Semantic Archival/Knowledge Base (SA/KB) component with information relevant to the verbal and non-verbal communication analysis results (speech acts, valence & arousal, FrameNet context), and (ii) to provide information about the current user state to the DM. The information about the user’s emotional state and input sentences is used by the underlying ontology-based reasoning mechanisms for decision making and generation of pertinent outputs, such as missing information or final reasoning results on the identified topic. This output serves as foundation to decide on the next system action for the DM.

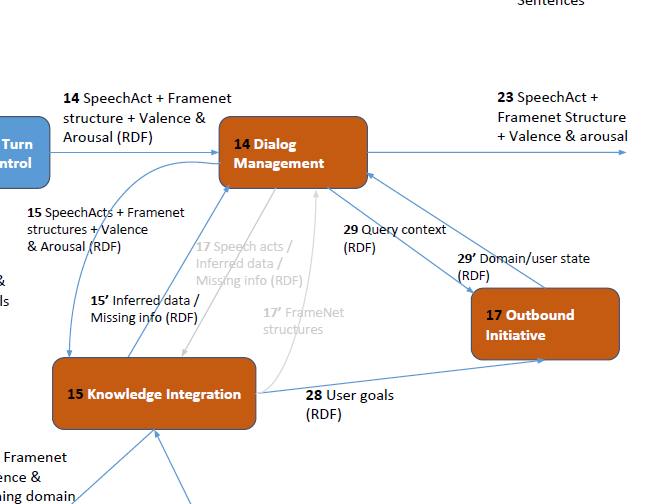


Figure 1 Overview of Dialog Management - Knowledge Integration interactions

## Dataflow 15 & 15’

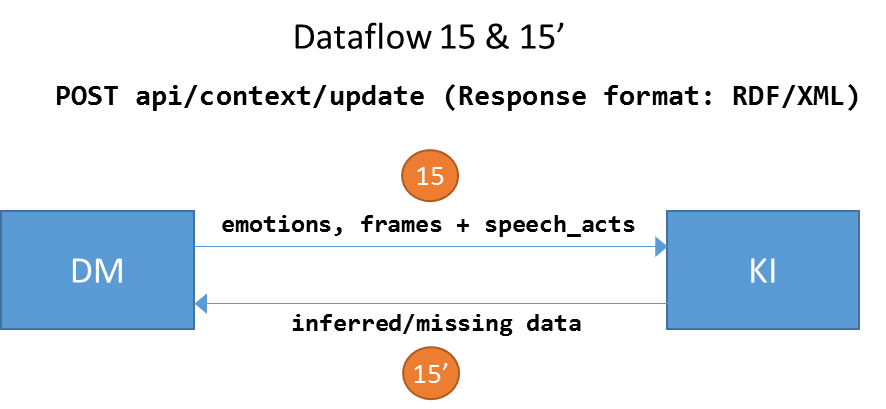
KI provides a REST API (HTTP POST) that can be used by DM to send data regarding speech acts, valence & arousal and FrameNet-related structures (all in RDF). It gets as response an ontology-based description with the analysis results that may also contain requests for additional input needed by the underlying reasoning mechanism.

**POST api/context/update (Response format: RDF/OWL)**

**Parameters**

emotions: RDF/OWL

frames (including speech acts): RDF/OWL



## Dataflow 17 & 17’

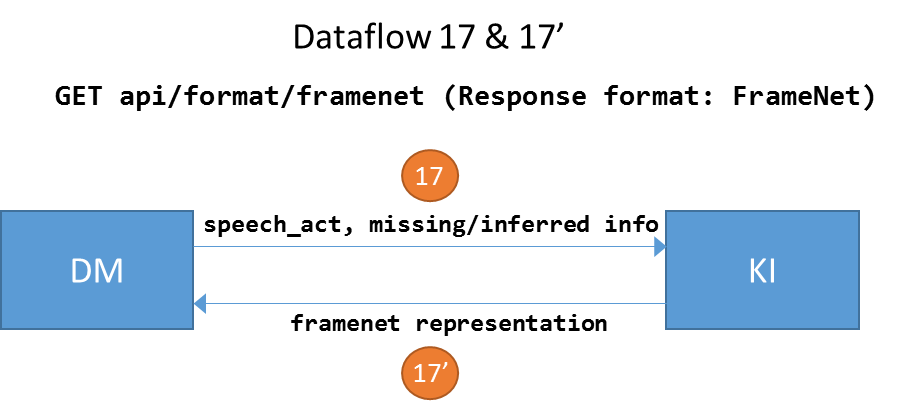
KI provides a REST endpoint (GET HTTP) that can be used by the DM to get a response that will be sent to the Spoken Language Generation component. As such, DM sends RDF/OWL data regarding speech acts and results of KI and gets as response the syntactically structured response in FrameNet that will be forwarded to the Spoken Language Generation component.

**GET api/format/framenet (Response format: FrameNet)**

**Parameters**

speech\_act: encoding RDF/OWL

info: encoding RDF/OWL



**Note:** After some discussions with UPF and UULM, it seems that the logic behind the api/format/framenet API (i.e. the OWL/RDF to FrameNet transformation) is better to be implemented by the component that actually needs FrameNet-based data (i.e. to be encapsulated inside the Spoken Language Generator component). That is why the pertinent interactions in Figure 1 have different color. This decision has been mainly based on the fact that this transformation requires some familiarity with the FrameNet and it should be driven by domain modeling and reasoning requirements of the component that is going to use this information, possibly reusing the work already done and the experience (know-how) gained from the implementation of the inverse transformation (FrameNet to OWL/RDF). However, in order to ease the integration, we have implemented the pertinent stub interface that can be used for testing, if needed.

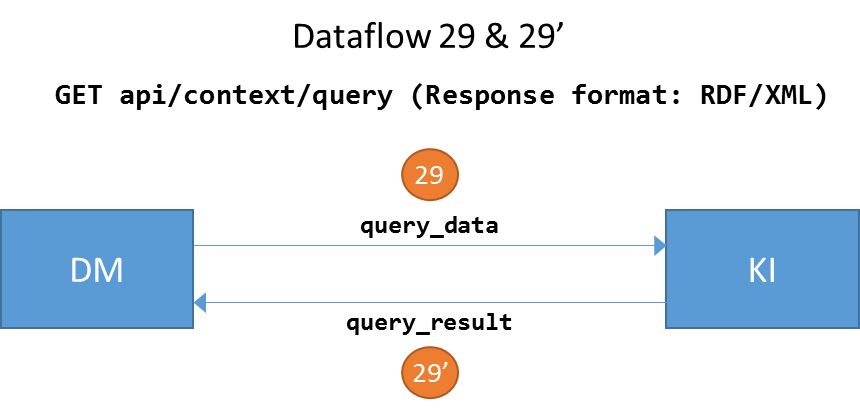
## Dataflow 29 & 29’

The aim of this interaction is to provide information about a requested topic or state, so as to trigger any alert and feedback mechanism, based on the information obtain from KI. To this end, KI provides a REST endpoint (GET HTTP) that van be used by the DM to query KI and get data for a specific topic by sending an RDF/OWL description of the requested query context.

**GET api/context/query (Response format: RDF/OWL)**

**Parameters**

query\_data: encoding RDF/OWL



## Knowledge Base / Semantic Archival

CERTH hosts the Knowledge Base of the architecture, which is a native RDF triple store. We have deployed an instance of the AllegroGraph RDFStore[[2]](#footnote-2) in an HP ProLiant server (Ubuntu Mate, Intel Xeon CPU E5-2620v3@2.40GHz, 1.5T HDD, 126GB RAM). The RDF store is accessible at http://160.40.51.155:10035, although there is no need to directly communicate with the triple store. This is done by KI. For integration and testing purposes, the api/context/update API has a dependency to the triple store. As such, each time this API is invoked, it opens a remote connection to the RDF triple store and inserts some dummy/fake data.

## Application Server

The developed REST APIs (JAVA 8 EE) have been deployed in an instance of the GlassFish server (v4.1.1 Community Edition) and can be directly invoked as:

POST http://160.40.51.155:8080/api/context/update

GET http://160.40.51.155:8080/api/context/query

GET http://160.40.51.155:8080/api/format/framenet

1. https://docs.google.com/presentation/d/1F36b5iKoLk6i60g6VJ\_JiMdlSF6xl1mTAoML8EZ6L\_M/edit?usp=sharing [↑](#footnote-ref-1)
2. http://franz.com/agraph/allegrograph/ [↑](#footnote-ref-2)