the application behavior according to the latest context"[38], but scalable solutions to do that in dynamic settings and going beyond complex event detection towards non-monotonic reasoning is still under investigation [39], [40].

The main novelty of this approach used by the *Contextual Filtering* component for adaptive and context-aware reasoning in dynamic environments is the ability to use such complex reasoning capabilities to efficiently and dynamically select only relevant information to tackle the challenge of converting data into knowledge in dynamic environments in a scalable way.

2) EVENT-BASED USER-CENTRIC DECISION SUPPORT

The *Decision support* component of the CityPulse framework represents *higher-level intelligence*, and the main role of this component is to enable reactive decision support functionalities to be easily deployed, providing the most suitable answers at the right time.

The reasoning capabilities needed to support users in making better decisions require handling incomplete, diverse and unreliable input, as well as constraints and preferences in the deduction process. This expressivity in the *Decision support* component is achieved by using a declarative non-monotonic logic reasoning approach based on Answer Set Programming. Semantic technologies for handling data streams, in fact, cannot exhibit complex reasoning capabilities such as the ability of managing defaults, common-sense, preferences, recursion, and non-determinism. Conversely, state-of-the-art logic-based non-monotonic reasoners can perform such tasks but are only suitable for data that changes in low volumes at low frequency. Therefore, the main challenge addressed by the *Decision support* component is to enable expressive reasoning for decision support in a scalable way. In our approach we combine the advantages of semantic query processing and non-monotonic reasoning based on the general idea behind StreamRule [41]. We have identified features that can affect scalability of such a combined approach and we have exploited the user-centric features and adaptive filtering of relevant events to reduce the input size and therefore the search space for the decision support task, thus increasing the potential for better scalability. As mentioned earlier in this paper, the user-centric and event-driven features of the *Decision support* component strongly rely on the tight interaction with the user application and the *Contextual Filtering* component respectively, with the former being in charge of the bidirectional communication with the other two, as represented in Figure 12.

In what follows, we detail the input and output used by the *Decision support* component, and also introduce the main reasoning modules that are currently implemented as part of the CityPulse framework. The input for the *Decision support* component is illustrated in Figure 13.

A Reasoning request consists of:

- User Reference: uniquely identifies the user that made the request. Such reference is related to user credentials that will be used in the final integration activities in order to manage user logins and instances of the CityPulse framework in different cities.
- Type: indicates the reasoning task required by the application. This is used directly by the *Decision support* component to select which set of rules to apply, and needs to be identified among a set of available options at design time by the application developer.
- Functional Details: represent the qualitative criteria used in the reasoning task to produce a solution that best fits the user needs.

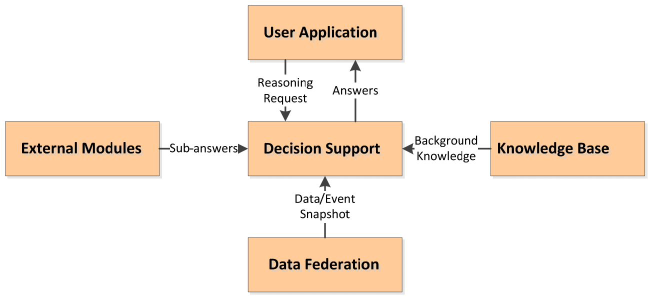


Figure 13. Decision Support I/O.

The Functional details are composed by:

- Functional Parameters, defining mandatory information for the Reasoning Request (such as start and end location in a travel planner scenario);
- Functional Constraints, defining a numerical threshold for specific functional aspects of the Reasoning Request (such as cost of a trip, distance or travel time in a travel planner scenario). These restrictions are evaluated as hard constraints, which needs to be fulfilled by each of the alternative solutions offered to the user;
- Functional Preferences, which encode two types of soft constraints: a qualitative optimisation statement defined on the same functional aspects used in the Functional Constraint (such as minimisation of the travel time); or a qualitative partial order over such optimization statements (such as preference on the minimisation of the distance over minimization of the travel time). Preferences are used by the Decision support component to provide to the user the optimal solution among those verifying the functional constraints.

Given to the fully declarative approach of ASP, we are able to provide a specification of all aspects of a Reasoning Request, which can be automatically mapped into logic rules. As a result, we achieve a high degree of flexibility to adapt to new scenarios and requirements, which makes it easier for developers to build new applications. In the reasoning process performed by the *Decision support* component, declarative rules derived from the Reasoning Request are combined in a single logic ASP program with the following input:

- Sub-Answers from external modules. These are facts computed by external models as subtasks of the decision support problem. Calls to such external modules can be used to improve scalability by reducing the solution space, or for privacy reasons. For example, in a travel planner scenario, a reduced list of all possible routes to go from A to B within a geographical area is provided by the *Geospatial Data Infrastructure* component. This does not imply any contextual reasoning, qualitative optimization, or event-driven adaptation, which is instead provided by the *Decision support* component.

- Background Knowledge. This is static information about a particular domain (such as the location of parking areas).
- Events Snapshot. This information comes from the *Data federation* component, and consists of the latest values of related events in the city (such as traffic levels in particular areas). Events snapshots are used the first time a solution is computed and ensures this solution is based on the most updated values. Dynamic changes to these values are then continuously detected and used by the *Decision support* component via the adaptive filtering mechanism of the *Contextual Filtering* component.

The *Decision support* component produces a set of answers to the Reasoning Request that satisfy all user's requirements and preferences in the best possible way. These solutions are computed by applying sets of rules deployed as scenario-driven decision support modules. We currently support three different types of decision support modules, covering a broad range of application scenarios:

- *Routing Module*. It provides the best solution(s) for a routing task, which is continuously updated based on incoming events and their criticality; the travel planner scenario relies on this module, and so do other scenarios where finding the optimal route is the main task, but the selection criteria (including constraints and preferences) might vary. Examples include the ability to schedule optimal pick-up for health services, green bike tours and alike.
- *Optimal Selection Module*. It provides the best selection among a set of alternative based on optimisation criteria, constraints and preferences; the parking scenario is part of this category.
- *Planning Module*. It provides optimal solutions to a planning problem, and continuously updates the options when the selected one is no longer feasible (based on Functional Constraints) or is no longer optimal (based on Functional Preferences); scenarios that are part of the cultural sector (such as planning activities in the city based on user interests or schedule) or the energy sector (such as planning household usage based on user-defined constraints and cost) are candidate scenarios for using this module.

These modules are available as API to be used by application developers in a broad range of applications. Since decision support tasks are strongly domain-dependent, different type of reasoning tasks would require additional Decision Support modules to be developed. The CityPulse framework provides a set of guidelines for developing new decision support modules, which requires knowledge of Answer Set Programming to develop the proper application logic.

3) TECHNICAL ADAPTATION

The CityPulse framework leverages the *Technical adaptation* component to automatically detect critical quality updates for the federated data streams used in the *Data federation* component and make adjustments. IoT streams are inherently dynamic in nature and often unreliable, hence more prone of getting fluctuations in the quality metrics. Therefore, it is utmost necessary to have a quality-aware adaptation mechanism for IoT streams. Using the *Technical adaptation* component, the quality of user queries deployed by the *Data federation* component can be maintained automatically by dynamically replacing data sources, unlike existing adaptive CEP systems which leverage query-rewriting and re-ordering of the query operators.

The CityPulse framework facilitates adaptability in quality-aware federation of IoT streams for smart city applications. In order to provide technical adaptation to increase robustness of smart city applications, following three steps are involved:

1. Monitoring Quality Updates: to monitor any updates in the quality metrics of the IoT streams involved in stream federation;
2. Evaluate Criticality: to determine whether any particular quality update is critical and if there is any adaptation action that should be carried out based on the composition plan and non-functional requirements; and
3. Adaptation Handling: when adaptation is triggered, determine the adaptation scope (i.e., part of the composition plan (produced by the *Data federation* component) that needs to be changed), make an adaptation request to the data federation and recompose the adaptation scope and redeploy the newly derived composition plan.

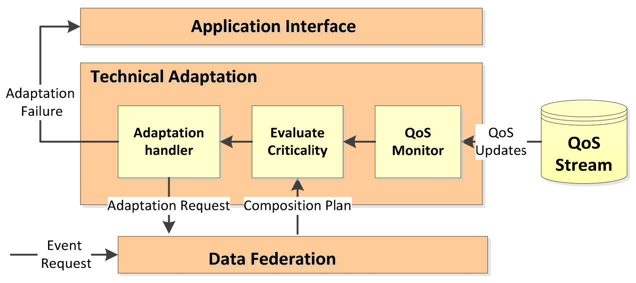


Figure 14. Adaptability in stream federation.

Figure 14 illustrates the process and the components used to achieve adaptability in stream federation following the above mentioned three steps.

During the event streams discovery and composition process, numerical quality vectors can be used to specify constraints over the quality metrics for data streams. For example, if we consider some typical quality attributes, including latency, price, energy consumption, bandwidth consumption, availability, completeness, accuracy and security, a numerical quality vector:

$$Q = \langle L, P, E, B, Ava, C, Acc, S \rangle$$

can be used to specify the quality of a data stream w.r.t. these dimensions along with the static data stream description. Similarly, a quality constraint vector Q' can be used to specify the quality constraints as thresholds for the quality metrics in the stream discovery or composition requests. If none of the quality values in Q breaks the threshold in Q', the data stream is considered to be a candidate for the discovery and composition. A weight vector:

$$W = \langle Lw, Pw, Ew, Bw, Avaw, Cw, Accw, Sw \rangle$$

contains 0 to 1 weights for each quality metrics within the requests, representing the preferences over quality metrics. The weight vector is used to normalize the quality vectors for all candidates based on simple additive weighting, the results are ranked and the top candidate is chosen. During runtime, the quality vectors will be re-computed based on most recent quality value updates, and the constraints are re-evaluated to determine if the quality of the data stream still satisfy the constraints. If optimal candidates must be used for the application domain at all times, updated quality vectors are also re-ranked, and the new top candidate is chosen to be the adaptation result. It is worth mentioning that such preferences and requirements are specified as a default configuration within a specific application, and they can be overwritten by user specific settings.

The *Technical adaptation* component is closely integrated with the *Data federation* component. It receives quality updates disseminated in the *Data Bus*. The quality updates are originally produced by the Quality Monitoring component. The *Technical adaptation* component is transparent to the end user as long as the adaptations are successful. Otherwise it may produce a failure notification to the application interface. The adaptation strategies of the *Technical adaptation* component can be configured by a 3rd party developer by changing the *AdaptationMode* parameter in the API offered by the *Data federation* component. Then, an initialization API is invoked at the backend system to create an *AdaptationManager* instance for the specific request.

C. COMPONENTS INTERDEPENDENCIES

The CityPulse components are highly flexible which allow multiple configurations of exploitation. In other words, the application developer can deploy only a subset of the components based on the requirements of the application which have to be developed.

There are interdependencies among the components of the framework and it is possible that the application developer will have to deploy also other CityPulse components in order to have a certain feature running.

Considering the out of the box installation, when the application developer simply deploys and configures a component, he has to deploy also the CityPulse components which are bellow the considered component in the architecture (see Figure 1). For other types of installations the application developer can replace one or several components with its own custom made modules. This is possible because the components are using REST and AMQP protocols to communicate.

IV. CONTEXT-AWARE REAL TIME TRAVEL PLANNER

In order to demonstrate how the CityPulse framework can be used to develop applications for smart cities and citizens, we have implemented a context-aware real time *Travel Planner* using the live data from the city of Aarhus, Denmark. The scenario was defined in collaboration with the IT departments of the Aarhus municipality and the following criteria have been considered: the impact of the application for the citizens and the data availability. The selected scenario belongs to the traffic domain, but as it was presented earlier in this paper the CityPulse framework is generic and it can be applied in any domain.

This scenario aims to provide travel-planning solutions, which go beyond the state of the art solutions by allowing users to provide multi dimensional requirements and preferences such as air quality, traffic conditions and parking availability. In this way the users receive parking and route recommendations based on the current context of the city. In addition to this, *Travel Planner* continuously monitors the user context and events detected on the planned route. User will be prompted to opt for a detour if the real time conditions on the planned journey do not meet the user specified criteria anymore.

In this case, the application developer has performed the following activities:

- Configured the CityPulse framework components;
- Deployed the components into a back end server;
- Developed a smart phone application using the APIs exposed by the framework in order to respond to the user requests. This application does not perform any data processing.

For this application, and very probably for most of the CityPulse enabled applications, the framework components can be divided in two different categories. First of all, the large scale data stream processing modules are configured and deployed in order to permanently monitor the status of the Aarhus city. In this way, when a new user logs into the mobile application and generates a request for a recommendation, the back-end application can provide the answer based on the current traffic or pollution situation from the city.

The second category of components, which perform real-time adaptive urban reasoning, are triggered when the user requests the routing or parking recommendation.

The following subsections present the specific workflows for the two categories of components mentioned above. For each particular workflow the activities, which have to be per-

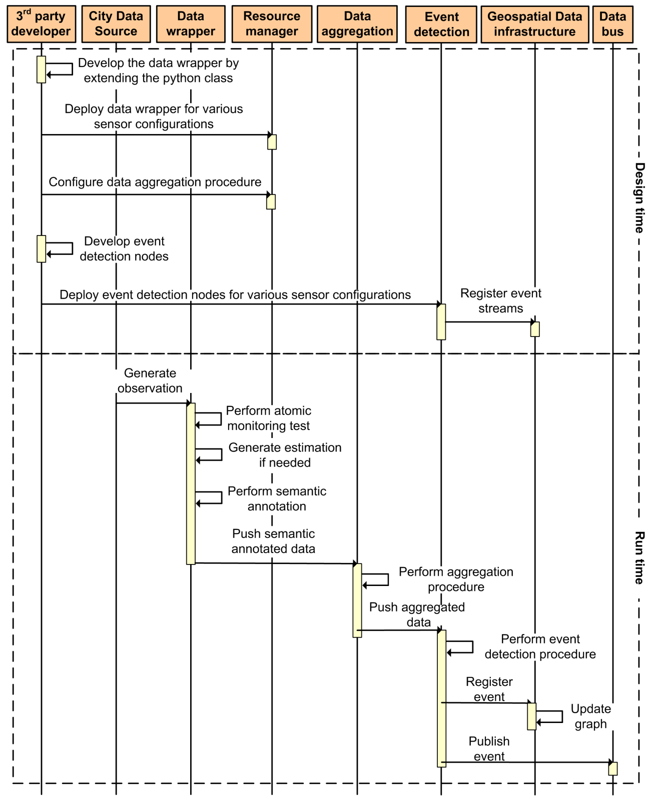

3rd party developer
City Data Source
Data wrapper
Resource manager
Data aggregation
Event detection
Geospatial Data infrastructure
Data bus
Develop the data wrapper by extending the python class
Deploy data wrapper for various sensor configurations
Configure data aggregation procedure
Develop event detection nodes
Deploy event detection nodes for various sensor configurations
Register event streams
Design time
Generate observation
Perform atomic monitoring test
Generate estimation if needed
Perform semantic annotation
Push semantic annotated data
Perform aggregation procedure
Push aggregated data
Perform event detection procedure
Register event
Update graph
Publish event
Run time

Figure 15. Large scale data stream processing workflow.

form by the application developer at design time, are marked at the beginning.

A. LARGE SCALE DATA STREAM PROCESSING WORKFLOW

For the considered scenario the large scale data stream processing modules are configured to process in real time the parking and traffic data coming from the city sensors with the scope of detecting relevant events for the users traveling in the city. Figure 15 depicts the workflow and this subsection is dedicated to explain the activities.

For the realization of the envisaged Travel Planner application the required data streams must be made available within the framework at first in the design phase. For this, corresponding *Data wrappers* for both data streams needed to be developed. The Aarhus Traffic wrapper was realised implementing a HTTP Pull Connection (i.e. HTTP client) extending the abstract Sensor Connection. It fetches new sensor readings from the ODAA portal. The response messages are JSON encoded documents. Consequently a JSON Parser was implemented to extract the relevant information out of these messages, namely the number of vehicles passing the two measurement points (''vehicleCount'') and their average speed (''avgSpeed'').

The second *Data wrapper* implemented fetches the parking data of parking garages in Aarhus. Similar to the traffic data stream the Aarhus Parking data is provided by the ODAA platform and encoded as JSON message. Therefore the same HTTP Pull Connection but a different JSON Parser is used. The stream provides information about the total number of parking spaces in the garage (''totalSpace'') and the number of occupied spaces/vehicles in the garage (''vehicleCount'').

```
insert into ParkingGarageStatusStream
select * from ParkingGarageStream.win:time(parkingMonitoringInterval sec)
having (max(ParkingGarageStream.
numberOfCars)-min(ParkingGarageStream.numberOfCars)/
ParkingGarageStream.parkingCapacity) > ocupancyChangeRateTreshold)
```

Listing 4. Detect parking occupancy rate events.

Both *Data Wrappers* were deployed in the *Resource management*. During run time new observations are fetched in a five minute interval for the Aarhus Traffic stream and in a one minute interval for the Aarhus Parking stream respectively.

The next step was to configure the *Data aggregation* component to use the SensorSAX algorithm as aggregation method for the traffic and parking observations. Due to the fact that it is a multi-resolution approach, it is triggered based on the variation in data stream, and can be configured by sensitivity level to instantly report any change to the system for further processing, such as *Event detection*.

The last step considered during the design time was to develop the Event detection nodes, which are used to process the Aarhus traffic and parking aggregated data streams with the scope of identifying relevant events from the end user perspective. In that sense, Event detection nodes have been developed and deployed in order to detect the traffic jams and the parking status changes. In the following paragraphs we will present the Event detection node used for parking places fast vacancy modification (in other words when a lot of vehicles enter the parking garage in a very short period of time).

The input of the event detection adapter is represented by one parking data stream and the configuration parameters are:

- *ocupancyChangeRateTreshold*: the rate of car entering into the garage in order to generate the event;
- *parkingMonitoringInterval*: the length of the time interval for which the occupancy change rate is computed.

The statement from Listing 4 represents the detection logic which has to be included into the parking Event Detection node in order to achieve the goal. First, it computes the minimum and the maximum number of cars which have been in the garage in the last *parkingMonitoringInterval* seconds. Then, the statement determines with what percentage the parking occupancy has been modified by dividing the difference between the maximum and the minimum values to the total capacity of the garage. If the percentage is bigger than the *ocupancyChangeRateTreshold*, then a parking place fast vacancy modification event is generated.

For the implementation of the Travel Planner application the domain experts do not need to make changes to the *Atomic monitoring* nor *Composite monitoring*. As the quality calculation follows an application independent approach, there is no need for any changes. The adaptations the Domain Expert has to do are limited to providing a description for newly deployed sensors (SensorDescription, see Listing 1)

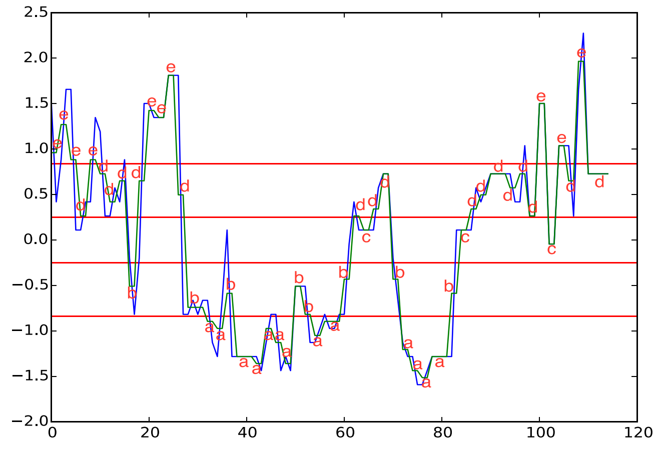


Figure 16. **Data aggregation using SensorSAX with minimum window size is 1 and sensitivity level is 0.3 for average speed observation of Aarhus traffic data stream.**

used by the application. In the case of the Travel Planner application SensorDescriptions for the traffic and the parking stream were provided. The introduced QoI Explorer enables the application developer to check the quality of sensors along a selected route and to check the results for plausibility.

At run time, after the CityPulse components have been deployed, the *Data wrappers* start to fetch observations from the Aarhus city data streams. When an observation is received the atomic monitoring is performed and the QoI description of the stream is updated. If the observation is missing or the quality of the stream is low, the *Fault recovery* component is triggered to generate an estimation.

The observations along with the QoI determined in the *Atomic monitoring* is annotated semantically afterwards. The annotation process is generic and uses the details provided in the SensorDescription, where required information such as the general domain of the stream; the nature/concept of an observation; and the unit of measurement for the observation are specified.

Next, the numeric values of the observation are aggregated according using the selected algorithm. Figure 16 depicts the data captured for average speed via the corresponding sensor points and illustrate SensorSAX patterns created from the raw data.

At the end the aggregated streams of observations are processed by the *Event detection* component in order to extract the parking and traffic events. Once an event is detected it is published on the *Data Bus*, to be further used by the *Decision support* and *Contextual Filtering* components and a notification is sent to the *Geospatial Data Infrastructure*. In this way, the process of computing the routes (see the Geospatial Data Infrastructure APIs from Section 3) is influences by the current city context.

B. REAL-TIME ADAPTIVE URBAN REASONING FOR TRAVEL PLANNER

For the considered scenario, the real time adaptive urban reasoning components are used to provide answers, when a user generates routes and parking recommendation requests.

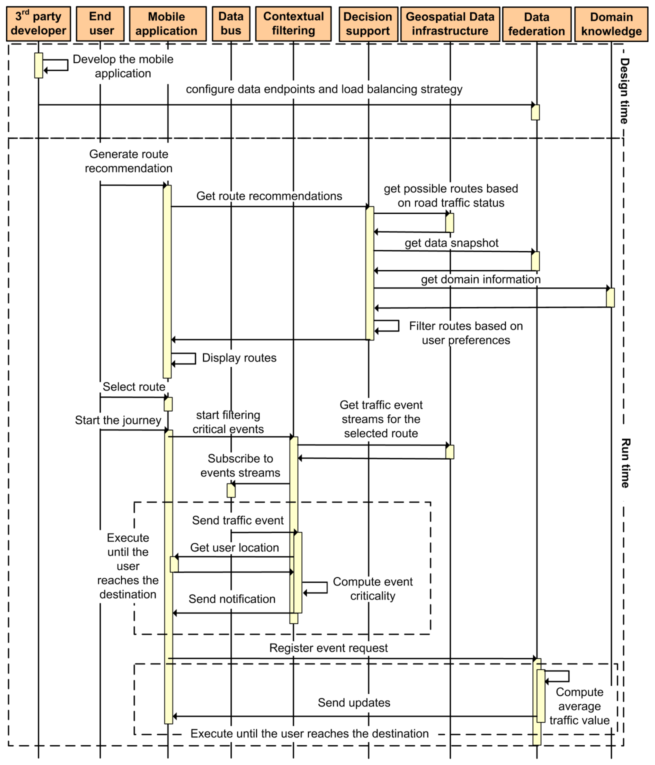

3rd party developer
End user
Mobile application
Data bus
Contextual filtering
Decision support
Geospatial Data infrastructure
Data federation
Domain knowledge
Design time
Run time
Develop the mobile application
configure data endpoints and load balancing strategy
Generate route recommendation
Get route recommendations
get possible routes based on road traffic status
get data snapshot
get domain information
Filter routes based on user preferences
Display routes
Select route
Start the journey
start filtering critical events
Get traffic event streams for the selected route
Subscribe to events streams
Send traffic event
Get user location
Execute until the user reaches the destination
Compute event criticality
Send notification
Register event request
Send updates
Compute average traffic value
Execute until the user reaches the destination

Figure 17. **Real-time Adaptive Urban Reasoning workflow for Travel Planner.**

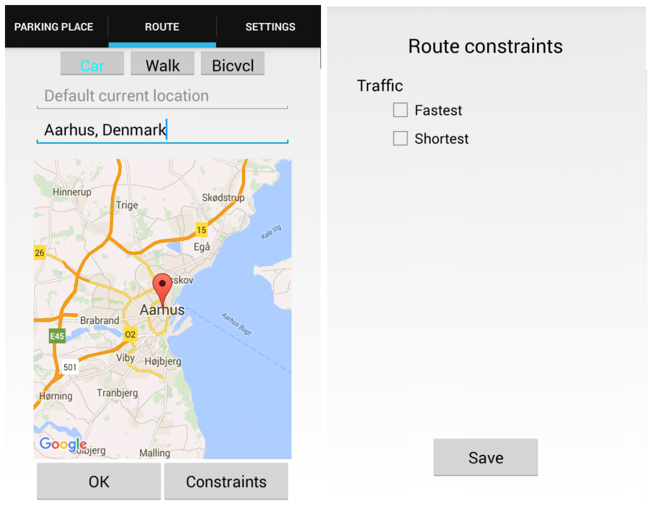
Figure 17 depicts the workflow executed by the components for providing an answer when route a recommendation is requested.

As mentioned above, all the CityPulse framework components are deployed on a back-end server and are accessible via a set of APIs. As a result of that the application developer has only to develop a user-friendly front-end application, which calls the framework APIs. In our case we have developed an Android application.

Figure 18 depicts the user interfaces used by the end user to set the travel preferences and the destination point.

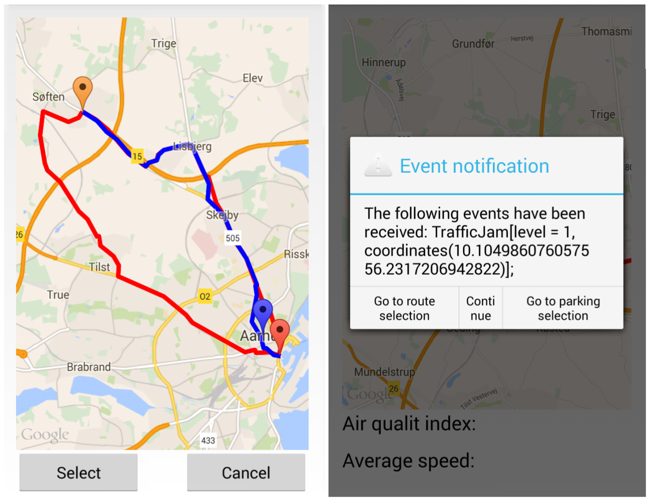
After the user has filled in the details and made the request using the user interface, the mobile application generates the appropriate request for the *Decision support* component which has the following main fields:

- Type: indicating what decision support module is to be used for this application ("TRAVEL-PLANNER" in this case);
- Functional details: specifying possible values of user's requirements, including:
  - Functional parameters: mandatory information that the user provides such as starting and ending locations, starting date and time, and transportation type (car, bicycle, or walk).
  - Functional constraints: numerical thresholds for cost of a trip, distance, or travel time.
  - Functional preferences: the user can specify his preferences along selected routes, which hold the



a) b)

Figure 18. The user interfaces of the Android application used to select the starting point (a) and the travel preferences (b).



The following events have been received: TrafficJam[level = 1, coordinates(10.1049860760575 56.2317206942822)];

a) b)

Figure 19. The user interfaces of the Android application: a) select the preferred route; b) notification of a traffic jam which appeared on the selected route while the user is travelling.

functional constraints. These preferences can be the minimization or the maximisation of travel time or distance.

The functional constraints and preferences specify different thresholds and minimisation criteria for electing the route. During the development of the mobile application the domain expert has computed a default set of values for these thresholds. As a result of that, the route constraints user interface from.

Figure 18 b) allows the user to select between the fastest/shortest routes. If needed, more fields can be added in this user interface in order to allow more fine-grained constraints specification, but the usability of the application may suffer.

```
1.  input_get_routes(SP, EP, V, 5) :-
     parameter("STARTING_POINT", SP),
     parameter("ENDING_POINT", EP), route_costMode(V).
2.  route(@get_routes(SP, EP, V, N)) :- input_get_routes(SP, EP, V, N).
3.  route_data(@get_routes_data(SP, EP, V, N)) :- input_get_routes(SP,
    EP, V, N).
4.  parameter("STARTING_POINT","10.116919 56.226144").
5.  parameter("ENDING_POINT","10.1591864 56.1481156").
6.  minimize{AV@2 : valueOf("DISTANCE", AV)}.
7.  minimize{AV@1 : valueOf("TRAVEL_TIME ", AV)}.
```

Listing 5. A snapshot of logic decision support rules for the Travel Planner scenario.

This concrete reasoning request is automatically mapped into ASP rules (see example rules 4-7 in Listing 5), and combined with the specific scenario-driven rules for the Travel Planner Decision Support module.[16] The *Decision support* component collects all possible routes from the *Geo-spatial database infrastructure* as well as the last snapshot of values of relevant functional properties for those routes which can be produced dynamically by the *Data federation* component or retrieved from the *Knowledge base* (rules 1-3).

Afterwards the Decision support component relies on the Clingo4 ASP reasoner to compute the optimal routes that satisfy the user's reasoning request best. In the current implementation of the Decision support, the user will receive at most 5 optimal routes.

Figure 19 a) depicts the user interface where the routes computed by the Decision support are displayed to the end user.

After the user selects the preferred route, a request is generated to the *Contextual filtering* component in order to identify the relevant events for the use while he/she is traveling. The request includes the following properties:

- Route of interest: the current route of the user
- Filtering Factors: used to filter unrelated (unwanted) events out, and include event's source, event's category, and user's activity, as specified in a user-context ontology.[17] For this particular scenario, the activities included in our ontology include CarCommute (user is traveling by a car), or Walk, or BikeCommute (user is traveling by a bike).
- Ranking Factor: identifies which metric is preferred by the user for ranking the criticality of incoming events. We have currently implemented the Ranking Factor based on two metrics: distance, and gravity of the event. In order to combine these two metrics, we use the linear combination approach, where the user can identify weights (or importance) for each metric.

Similar to the decision support constraints and preferences, the filtering and ranking factors are selected by the domain

[16]Note that this can be fully automated due to the declarative nature of our approach to rule-based reasoning based on ASP.

[17]Note that the ontology can be extended by adding new activities.

expert during the mobile application development stage, but they can be made accessible to the end users.

Once the user has selected one of the routes computed by the Decision support component, the Contextual Filtering component sends a request to the Geospatial Database Infrastructure component to obtain the description of the event streams available on the selected route, as registered at design time by the Event detection component. The Contextual Filtering component uses these descriptions to subscribe to detected events via a Data Bus. In addition, the Contextual Filtering also receives the contextual information of the user (currently including location) from the user/application as a stream. Whenever there is a new event detected by the Event detection component, the Contextual Filtering component filters and assigns the most appropriate criticality (0 if not critical, from 1 to 5 if it is critical) to the new event. If the new event is marked as critical,[18] the user receives a notification and he/she has the option to change the current solution and request a new one or ignore the event.

Figure 19 b) depicts the notification received by the end user, while s/he is traveling and a traffic event is detected on his/hers route.

In addition to the contextual filtering request, the mobile application triggers the *Data federation* to continuously compute the average speed of the cars from the selected route. At design time, the application developer has configured the *Data federation* component to store the meta-data for the traffic sensors and to use the "EL" load balancing strategy (starts with one engine instance and creates more instances elastically when all existing instances have reached maximum capacity).

The request generated by the mobile application contains the following fields:

- ep: the query pattern, in this application it is a conjunction of primitive traffic report events,
- constraint: the QoS constraint vector, absence of the constraint results in application of a set of defaulted loose constraints,
- weight: the QoS weight vector, absence of the weights results in equal weights configured to all QoS metrics,
- continuous: true have been selected to compute the average continuously;
- engineType: type of RDF processing engine to be used, can be 'CQELS' or 'CSPARQL'
- aggOp: aggregation operator, which for our particular situation is average.

The *ep* in the request contains the functional requirements for the primitive traffic data streams, e.g. what properties should they measure and what are the locations of the sensors (computed from the route selected by the user). Combining the functional requirements with the QoS constraints and preferences, the *Data federation* component creates the optimal composition plan for the request based on the stream meta-data provided in the knowledge based.

[18]Note that we currently provide all events marked with criticality higher than 0, but this can be changed by fixing a different threshold or limiting the notification to the top-k events.

```
SELECT ?obId1 ?obId2 ?v1 ?v2
WHERE { ?p1   a ct:AverageSpeed.
          ?p2   a ct:AverageSpeed.
STREAM <Traffic226> [range 3s ]
          {?obId1 a ?ob.
            ?obId1 ssn:observedProperty ?p1.
            ?obId1 sao:value ?v1.}
STREAM <Traffic439> [range 3s ]
          {?obId2 a ?ob.
            ?obId2 ssn:observedProperty ?p2.
            ?obId2 sao:value ?v2.}
}
```

Listing 6. Sample CQELS query generated from composition plan.

According to the *engineType* parameter specified in the request, the composition plan is transformed into a CQELS or CSPARQL query, as shown in Listing 6. A post-processing is applied to the query evaluation results to aggregate the observation values, using the aggregation operator specified in the *aggOp*.If the *aggOp* is set to empty then no post-processing is invoked.

The Data Federation is a generic component that gives continuous query results over federated data streams. Since it follows a service-oriented approach, the discovery and composition algorithms do not require changes based on specific application domains, as long as the service description model is used by the service providers and consumers.

The domain experts can choose between different default target continuous query evaluation systems (currently CQELS and C-SPARQL are integrated), based on the different characteristics of the application domains, e.g., the average query size, the frequency of data streams and the size of the background knowledge. These factors may affect the performance of the target systems (for more information refer to [42]) and a configuration based on specific scenario could be beneficial, but it is not mandatory.

V. CONCLUSIONS AND FUTURE WORK

Providing enhanced services to citizens while cities are growing due to urbanisation, and while resources are limited demands for a more intelligent use of the existing resources. The cities have started to deploy sensor and actor devices in their environment, e.g. intelligent lighting, and observation and monitoring devices to collect traffic, air quality and water/waste data. However, the current focus is mainly on collection, storage and visualisation of the datasets with an emphasis on high performance computing and visual computing solutions. While the recent efforts in this area have enabled emerging technologies and solutions to develop novel techniques for smart city applications and use-cases scenarios, there is however a gap in providing efficient and scalable methods that enable (near)real-time processing and interpretation of streamed sensory and social media data in smart city environments.

This paper proposes a framework for large-scale data analytics to provide information in (near-)real-time, transform raw data into actionable information, and to enable creating ‘‘up-to-date’’ smart city applications. To deal with the heterogeneity of the datasets a virtualisation technique is employed using *Data Wrappers*. However raw data will not be directly machine-readable and it hinder automated interpretation of the collected data. In our work, the datasets are semantically annotated that enable interoperability and provide machine-readable/interpretable representation of the data streams. The varying quality of the data is considered from the beginning by providing quality measures. Data stream aggregation and fault recovery techniques are used to enhance the quality-aware access and processing of the data streams. To extract events from the large data sets in (near-)real-time, complex event processing and contextual filtering methods are used.

The proposed framework has been demonstrated in the paper by a smart Travel Planer application. The main contributions of this work include integrating of heterogeneous data streams, providing interoperability, quality analysis, (near-) real-time data analytics and application development in a scalable framework. The CityPulse components have been developed as reusable entities and are provided as open-source software that are available via the CityPulse github repository (https://github.com/CityPulse).

The CityPulse middleware components are also reusable in different application domains and are provided as open-source.

In order to reduce complexity and time for developing new applications a set of APIs is provided by each of the CityPulse components. This way the developers of services are able to abstract the complexity of the CityPulse middleware and are not bound to use specific technologies for the implementation.

The future work will focus on evaluation of the proposed framework for (near-)real-time city data analytics in different domains. The framework will be also used to provide data access user interfaces and prototype applications for smart city use-cases in the city of Aarhus and the city of Brasov.

REFERENCES

[1] S. Beswick, ‘‘Smart cities in Europe enabling innovation,’’ Osborne Clarke, London, U.K., Tech. Rep., 2014. [Online]. Available: http://www.cleanenergypipeline.com/Resources/CE/ResearchReports/Smart%20cities%20in%20Europe.pdf

[2] M. Naphade, G. Banavar, C. Harrison, J. Paraszczak, and R. Morris, ‘‘Smarter cities and their innovation challenges,’’ *Computer*, vol. 44, no. 6, pp. 32–39, Jun. 2011.

[3] T. W. Mills. (Dec. 2015). *Intel Corporation—Intel Labs Europe: Open Innovation 2.0*. [Online]. Available: http://dspace.mit.edu/handle/1721.1/99033

[4] F. Lécué, R. Tucker, V. Bicer, P. Tommasi, S. Tallevi-Diotallevi, and M. Sbodio, ‘‘Predicting severity of road traffic congestion using semantic Web technologies,’’ in *Proc. ESWC*, Crete, Greece, 2014, pp. 611–627.

[5] iCity Consortium. (Jun. 11, 2014). *iCity Project*. [Online]. Available: http://www.icityproject.eu/

[6] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs, ‘‘Building a big data platform for smart cities: Experience and lessons from Santander,’’ in *Proc. IEEE Int. Congr. Big Data*, New York, NY, USA, Jun./Jul. 2015, pp. 592–599.

[7] J. Soldatos *et al.*, ‘‘OpenIoT: Open source Internet-of-Things in the cloud,’’ in *Interoperability and Open-Source Solutions for the Internet of Things*, I. P. Žarko, K. Pripužić, M. Serrano, Eds. Berlin, Germany: Springer, 2014, pp. 13–25.

[8] D. Pfisterer *et al.*, ‘‘SPITFIRE: Toward a semantic Web of things,’’ *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 40–48, Nov. 2011.

[9] R. Giaffreda, ‘‘iCore: A cognitive management framework for the Internet of Things,’’ in *The Future Internet*, A. Galis and A. Gavras, Eds. Heidelberg, Germany: Springer, 2013, pp. 350–352.

[10] R. Stühmer, Y. Verginadis, I. Alshabani, T. Morsellino, and A. Aversa, ‘‘PLAY: Semantics-based event marketplace,’’ in *Proc. IFIP Working Conf. Virtual Enterprise-Special Session Event-Driven Collaborative Netw.*, 2013, pp. 699–707.

[11] F. Lécué *et al.*, ‘‘Smart traffic analytics in the semantic Web with STAR-CITY: Scenarios, system and lessons learned in Dublin City,’’ *Web Semantics, Sci., Services Agents World Wide Web*, vols. 27–28, pp. 26–33, Aug./Oct. 2014.

[12] Z. Zhao, W. Ding, J. Wang, and Y. Han, ‘‘A hybrid processing system for large-scale traffic sensor data,’’ *IEEE Access*, vol. 3, pp. 2341–2351, Nov. 2015.

[13] J.-P. Calbimonte, S. Sarni, J. Eberle, and K. Aberer, ‘‘XGSN: An open-source semantic sensing middleware for the Web of things,’’ in *Proc. 7th Int. Conf. Semantic Sensor Netw.*, Riva Del Garda, Italy, 2014, pp. 1–6.

[14] O. Fambon, É. Fleury, G. Harter, R. Pissard-Gibollet, and F. Saint-Marcel, ‘‘FIT IoT-LAB tutorial: Hands-on practice with a very large scale testbed tool for the Internet of Things,’’ in *Proc. UbiMob*, 2014, pp. 1–5.

[15] *NYC BigApps 2015*, accessed on Dec. 20, 2015. [Online]. Available: http://bigapps.nyc/

[16] Z. Khan, A. Anjum, K. Soomro, and M. A. Tahir, ‘‘Towards cloud based big data analytics for smart future cities,’’ *J. Cloud Comput.*, vol. 4, p. 2, Dec. 2015.

[17] *Amsterdam Smart City*, accessed on Dec. 20, 2015. [Online]. Available: http://amsterdamsmartcity.com/#/nl/home

[18] *AMQP Specification*, accessed on Dec. 20, 2015. [Online]. Available: http://www.amqp.org/specification/1.0/amqp-org-download

[19] R. Agrawal, C. Faloutsos, and A. Swami, ‘‘Efficient similarity search in sequence databases,’’ in *Foundations of Data Organization and Algorithms*. Heidelberg, Germany: Springer, 1993.

[20] A. Haar, ‘‘Zur theorie der orthogonalen funktionensysteme,’’ *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, 1910.

[21] Z. R. Struzik and A. Siebes, ‘‘The Haar wavelet transform in the time series similarity paradigm,’’ in *Principles of Data Mining and Knowledge Discovery*. Berlin, Germany: Springer, 1999, pp. 12–22.

[22] K. Chakrabarti and S. Mehrotra, ‘‘Local dimensionality reduction: A new approach to indexing high dimensional spaces,’’ in *Proc. VLDB*, 2000, pp. 89–100.

[23] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, ‘‘Dimensionality reduction for fast similarity search in large time series databases,’’ *Knowl. Inf. Syst.*, vol. 3, no. 3, pp. 263–286, 2001.

[24] T. Berners-Lee, J. Hendler, and O. Lassila, ‘‘The semantic Web,’’ *Sci. Amer.*, vol. 284, no. 5, pp. 28–37, May 2001.

[25] M. P. Papazoglou, ‘‘Service-oriented computing: Concepts, characteristics and directions,’’ in *Proc. 4th Int. Conf. Web Inf. Syst. Eng. (WISE)*, Washington, DC, USA, Dec. 2003, pp. 3–12.

[26] D. C. Luckham, *The Power of Events*. Reading, MA, USA: Addison-Wesley, 2002.

[27] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, ‘‘C-SPARQL: SPARQL for continuous querying,’’ in *Proc. 18th Int. Conf. World Wide Web*, 2009, pp. 1061–1062.

[28] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth, ‘‘A native and adaptive approach for unified processing of linked streams and linked data,’’ in *Proc. 10th Int. Semantiv Web Conf. (ISWC)*, Bonn, Germany, Oct. 2011, pp. 1–16.

[29] F. Gao, E. Curry, and S. Bhiri, ‘‘Complex event service provision and composition based on event pattern matchmaking,’’ in *Proc. 8th ACM Int. Conf. Distrib. Event-Based Syst.*, Mumbai, India, 2014, pp. 71–82.