GRANT AGREEMENT No 609035
FP7-SMARTCITIES-2013

Real-Time IoT Stream Processing and Large-scale Data Analytics for Smart City Applications

Collaborative Project

Real-time Adaptive Urban Reasoning

Document Ref.	D5.1
Document Type	Report
Workpackage	WP5
Lead Contractor	NUIG
Author(s)	Alessandra Mileo, Feng Gao, Muhammad Intizar Ali, Pham Le Thi Anh Thu, Maria Bermudez, Daniel Puschmann
Contributing Partners	NUIG, UNIS, WSU
Planned Delivery Date	M12
Actual Delivery Date	31st July 2014
Dissemination Level	Public
Status	Completed
Version	V0.8-Final
Reviewed by	Dan Puiu (SIE), Daniel Kuemper (UASO), Payam Barnaghi (UNIS)

Executive Summary

CityPulse operates in dynamic smart city environments in which the properties of underlying resources and streams need to be constantly updated according to changes and events in the real world. In most of the existing solutions matchmaking between the requirements expressed by businesses and applications and available data is carried out at design-time. This approach is often far from optimal and its deficiencies become even more obvious in IoT scenarios where the properties of underlying services and resources dynamically change and depend on physical world events and phenomena (e.g. sensor readings, network availability, weather conditions, and temperature). In this deliverable we detail the CityPulse solution for adaptive data processing. We describe the development of efficient methodologies that control what is processed in the fast big data analytics pipeline and explain how to use the knowledge available at the higher-level for interpretation. The components that enable adaptive urban reasoning use the semantic data streams from the lower-layers and adaptively control the configuration of the lower-data processing pipelines, namely the discovery and selection process that continuously identifies the "best" streaming sources based on a set of technical as well as application- and user- dependent requirements. The Adaptive process in CityPulse uses domain knowledge and event correlation to determine the contextual relevance of higher-level events (i.e. machine interpretable or human understandable events) and adapt the DS process accordingly.

Table of contents

List of Figures

Abbreviations List

ASP	Answer Set Programming
CF	Contextual Filtering
CQELS	Continuous Query Execution over Linked Streams
DS	Decision Support
EBN	Event Broker Network
FPs	Functional Properties
GA	Genetic Algorithm
LOD	Linked Open Data
LODE	Linked Open Description of Events
LSM	Linked Stream Middleware
NFPs	Non-Functional Properties
QoS	Quality of Service
QoI	Quality of Information
TA	Technical Adaptation
WP	Work Package

1. Introduction

This deliverable describes the mechanism for real-time adaptive urban reasoning, which is mainly concerned with providing a higher-level control to i) configure data streaming in terms of discovery, federation and compositions of event streams performed in WP3, and ii) identify unexpected events that are contextually relevant for the DS functionality in WP5.

In this document we provide both the conceptualization and the implementation of a set of methods that can integrate adaptive behavior with real-time processing components for efficient and scalable adaptive urban reasoning. In order to achieve this, we will combine, in a pipeline, efficient stream query processing for detecting relevant streams, and (declarative) expressive stream reasoning tools to consume these events and provide alternative (context-aware) feedback for dynamic decision support. One benefit of a framework that enables this on-the-fly approach is that it enables detected events to be used in the reasoning process. Computed results are instantaneously presented to end-users or to an application that needs to react and support fast decisions in changing environments.

The remainder of this deliverable is organized as follows: Section 2 will describe the requirements and state-of-the-art on reactivity and adaptive functionalities in stream reasoning systems. Section 3 will provide an overview of the objectives and the approach for Real-Time Adaptive Urban Reasoning and introduce the concept of unexpected events, detailing the two separate control loops deployed for both low- and high- level adaptive behaviour. Section 4 will discuss more in details the user-dependent aspects and the way user profiles and requirements are taken into account in this process, while Section 5 will provide some details regarding the implementation of the components. We summarize the outcome of this activity as well as on-going work and future steps in Section 6.

2. Requirements and State-of-the-art

CityPulse provides solutions for adaptive data processing and utilisation and will enable knowledge-based automated composition of data/streams and resources using linked Cyber-Physical-Social resources. Using state-of-the-art semantic Web technologies, the technical work within WP3 will enable semantic transformation of the lower-level dynamic information (e.g., changes in sensor readings) to higher-level contextual abstractions (e.g. the occurrence of an accident or traffic jam). The interpretation of such semantic data as higher-level events is part of the event detection activity on WP3, but the interpretation of such events as relevant in a particular context is challenged by the changing nature of real-time demands of the targeted smart city applications.

Providing adaptive functionalities in this setting requires the development of efficient methodologies that can monitor events, detect (relevant) changes in real-time, and automatically generate a control loop in the fast big data analytics pipe for (re-) interpretation of decision support or reconfiguration of the IoT stream processing capabilities.

In the remainder of this section we provide some insights on requirements and state-of-the-art for these two directions of adaptation: reaction to changes in the way data is discovered, processed and aggregated as event services on one hand, and reaction to changes represented by high-level events having an impact on the results of an on-going decision support task on the other hand.

2.1 Adaptability in Stream Federation

Existing approaches to IoT stream processing address different issues related to streams and data processing in streams such as data management, query processing, and data mining. However, it is still an open challenge to properly address issues that are more closely related to the quality of a federated stream, more precisely integration of the federated data streams while keeping their quality metrics in mind.

Requirements

Quality metrics associated with the data streams can be defined as part of the user or application requirements, we refer these quality requirements (constraints and preferences) from users or applications as to Non-Functional Properties (NFPs). IoT streams are inherently dynamic in nature [23] and somehow unreliable, therefore more prone of getting fluctuations in the quality metrics. This is utmost necessary to have a quality-aware adaptation mechanism for (federated) IoT streams.

To facilitate adaptability in quality-aware federation of IoT streams for smart city applications, the following requirements have to be considered:

(i) Monitoring Quality Updates: to monitor any updates in the quality metrics of the IoT streams involved in stream federation,

(ii) Evaluate Criticality: to determine whether any particular quality update is critical and if there is any adaptation action that should be carried out, and

(iii) Technical Adaptation: Once the criticality of update is defined, perform necessary actions for automatic adaptation according to the new conditions/environment.

Deliverable D3.1 provided as outcome of WP3 semantically annotates the quality of data streams based on the quality metrics identified by WP4. It can be used to monitor the quality updates address (requirement (i) above) by telling the adaptation system which quality metrics are annotated (and thus could be monitored) and what are their semantics. For a specific instance of the adaptation system initialized by an event request, the set of quality metrics it monitors is determined by the event request: if the request specifies a constraint over a quality metric, this metric should be monitored. For example, if the application needs to obtain temperature readings for particular areas with accuracy greater than a particular value, this requirement will be part of the description of the event request (give me temperature reading for the specific area), therefore the adaptation component will monitor accuracy only for this particular event request. WP4 provides a monitoring mechanism for the quality of data streams, which also can be used to address requirement (i) by telling the adaptation system how to subscribe to the quality updates streams (via their semantic description) and receive notifications.

Future deliverable D3.2 provided as outcome of WP3 will detail the CityPulse mechanisms for data stream federation under functional and non-functional requirements. Outcome of stream federation algorithms described in D3.2 will give information of how a federated data stream is composed using other streams, and it can be used to address requirement (ii) by telling the adaptation system which federated streams are affected by the quality update of a specific stream and how. The federation components described in D3.2 can be used to address requirement (iii) by responding to the adaptation actions carried out by the adaptation system to create and deploy new stream federations under new conditions or environment changes.

State-Of-The-Art

Among various event publish/subscribe system architectures, Event Broker Networks (EBN) has gain much research interests in the last decade because of its scalability. Early works in EBN can be found in [11,34,35]. However, these systems have very different characteristics and they do not provide a common ground for comparing and evaluating them in terms of service quality [5].

A survey in [30] reviews a selection of eight EBN middlewares and analyses their support for five QoS metrics: latency, bandwidth, reliability, delivery semantics and message ordering. According to the survey, the support for QoS-aware event routing is quite poor: three of the eight middleware do not provide any support for the five metrics. Jedi [13] supports message ordering, Gryphon [7] supports delivery semantics, Hermes [34], Medym [9] and AR [29] supports reliability, IndiQoS [10] supports latency and bandwidth. The above QoS-aware EBNs (and many other peer works) develop various routing policies to determine event distribution routes with the optimal performance at run-time. However, they only support syntactical event subscriptions/publications. Also, most of them do not cater for complex events and event compositions. In CityPulse, complex event processing engines are incorporated in event brokers and provided as reusable semantic event services (or service compositions).

To facilitate quality-aware and adaptive service composition, the most prominent challenge is to compute optimal service compositions efficiently (related to requirement (iii) above), so that the run-time adaptation can be accomplished within acceptable delay. Various Genetic Algorithm (GA) based approaches, e.g., [43,8,42,17], have been studied to deal with this challenge. The above GA based approaches can only evaluate service composition plans with fixed sets of service tasks (abstract services) and cannot evaluate composition plans which are semantically equivalent but consists of service tasks on different granularity levels. A more recent work in [41] addresses this issue by presenting the concept of Generalized Component Services (GCS). However, [41] only caters for Input, Output, Precondition and Effect (IOPE) based service compositions. Complex event service composition requires an event pattern based reuse mechanism [18].

In CityPulse, we developed a novel GA for semantic event services, which creates quality-aware event service compositions efficiently to facilitate automatic adaptation actions for quality issues [Gao14-2]. This algorithm will be detailed in future deliverable D3.2, while the TA capability strictly relies on the ability to describe the QoI/QoS for semantic event services in order to monitor relevant changes and react to those by triggering new efficient semantic event services composition plans.

2.2 Adaptability in Knowledge Extraction and Decision Support

In this subsection we identify key requirements and state-of-the-art techniques for adaptability in knowledge extraction and decision support.

Requirements

Current techniques for knowledge extraction via stream reasoning do not cater for transforming IoT into actionable knowledge due to the lack of proper treatment of uncertainty (e.g. possible reasons of traffic jam vs. most probable reason of traffic jam) in the IoT environment.

The adaptive approach for knowledge extraction (represented by the Contextual Filtering component) provides a unified treatment of background knowledge and of uncertainty in the domain. The heterogeneity in the data streams is exploited to obtain complementary views of an event and estimate its criticality. For example, if camera sensors record a traffic jam in an area, and a user has already gone passed that area along a path provided by the DS component, the traffic jam is not going to be relevant for that user, but if the user changes his way for any reason, this can also be triggered as a event that requires to compute a new path to the destination, and this might results in the traffic jam becoming relevant. Correlations that exist between observations and events across different modalities are multidimensional and they are found using background knowledge and information theoretic approaches.

The requirements for an adaptive approach to knowledge extraction, can be associated to three steps, namely:

a) Unexpected events extraction: this step requires the identification of events that have an impact on the decision outcome and might require adaptation,
b) Estimation of the criticality of an event: this step requires to provide a quantitative analysis of the relationships between the unexpected event and the user contexts, to assess the potential impact of such unexpected event on the reasoning outcome, and
c) Selection of events to be filtered as critical to trigger a correspondent adaptation strategy: this step requires defining a cost function and using it to select which event requires adaptation of the reasoning process.

Following the above-mentioned steps the Contextual Filtering Component identifies critical events that can potentially affect the results of the DS Component.

During the pre-processing phase (a), WP3 is in charge of generating a list of unexpected events via pattern analysis from streaming data. The DS component subscribes to a subset of these events that are relevant from a specific DS module as specified at design time. The event detection component (from WP3, Activity A3.4) is triggered independently from a specific user request (e.g. a request from a smart city application) to aggregate sensor streams and detect unexpected events via pattern analysis. When the pattern is fulfilled an event that signals an unexpected situation (represented by a change in the real world) is sent to the Contextual filtering component.

This however, doesn't mean the event is critical.

The approach to tackle the event graph processing step (b) is an ontology-based approach used to automatically create relations between events. A dynamic event graph is created where unexpected events from WP3 as well as contextual elements (derived from contextual information) from DS are considered as graph nodes, while the relationship between an unexpected event and a contextual element is denoted by edges of the event graph. Each relation (edge) is assigned weight reflecting the level of criticality, and the graph structure is used to facilitate the computation of the level of influence of unexpected event over a contextual element.

In step (c), an overall contextual evaluation via a cost function is computed. A threshold of criticality level can also be defined for critical events. Using this threshold, the Contextual Filtering can successfully reduce the number of unexpected events that can help the DS to effectively perform reasoning tasks.

A detailed description of our approach to address requirements associated to steps (a), (b) and (c) is described in Section 3.2.

The ability to understand events from social media and adapt the reasoning process accordingly relies on building and populating an Event Ontology based on e.g. Linked Open Description of Events (LODE)[1] with City Events. The Event Ontology will be extended as new application areas are introduced. This Ontology will provide the decision makers with insights of knowledge structures, aiding them in better decision-making.

State of the art

Existing work in the domain of knowledge extraction and DS is mainly focused on the analysis of historical data. Many sophisticated methods and techniques have been already introduced, which can extract relevant information from huge amount of (mostly static) data and provide assistance for DS systems using extracted knowledge [15]. Some developments in the text analysis and mining realm have also been applied to the automatic extraction of knowledge from dynamic environments, for example in knowledge extraction from Web documents [Alani03]. However, stream processing applications requirements present different challenges because of the changing nature of the environment and the need to extract and convert information into actionable knowledge in (near) real-time.

In order to deal knowledge extraction in changing environments, research on context-aware systems focuses on the context-dependent nature of information, so that only relevant knowledge can be considered over time when the context changes. A well designed model is key for any context-aware system, and the literature contains a rich variety of context models for different purposes [3,38].

Context-awareness has been long studied in the domain of service provision and mobile computing. Context Aware applications are defined as “applications that can automatically adapt to the discovered context (environmental situations or conditions) by changing the applications behaviour according to the latest context”[12].

In terms of scalability of efficient solution for extracting knowledge in dynamic domains, [22] discussed the big data challenges in the domain of live data streams (web of things) and identified that biggest challenge for the applications consuming live data streams is not only huge volume of the data but also converting this huge data into actionable knowledge within a reasonable time is a challenging task. Various data pre-processing techniques e.g. aggregation, classification and clustering have been introduced to minimise the amount of data produced by the data streams [16,39] and to detect (complex) events in real-time [19].

[1] http://linkedevents.org/ontology/

The Contextual Filtering (CF) component developed in CityPulse leverages knowledge about the context in which data streams are produced and aggregated (including user context but also property of the data and of the reasoning task that will use it) in order to adaptively select what knowledge about streaming data (detected events in our case) is more relevant in a particular context. This not only requires to detect relevant events, but also relies on the ability to evaluate the criticality of those events based on the context.

Since we are operating in dynamic environment and streaming data are produces asynchronously, events can also be generated asynchronously at any time and require context-aware reasoning to be reactive to changes. Furthermore, given the potential incompleteness of available knowledge related to a particular changing context and related events, the ability to use common-sense and default reasoning, and to use preferences and optimization criteria to select the most plausible answers is also a key aspect for context-aware reasoning in streaming environments.

In the area of Knowledge Representation and Reasoning, several solutions have been proposed to deal with these aspects of knowledge during inference. Rule-based approaches and non-monotonic reasoning techniques have potential impact in a variety of real-world applications, but the ability of dealing with incomplete and noisy input streams, conflicts, defaults, qualitative preferences, constraints, non-determinism and generation of multiple solutions is computationally intensive. Extensions of Datalog towards the logic paradigm of Answer Set Programming (ASP)[1] have been implementing these reasoning capabilities, which can go far beyond the capabilities of existing query engines [31,37]. Logic programming dialects like Datalog with negation, covered by ASP, are viewed as a natural basis for the Semantic Web rule layer [36], but the full expressivity of ASP introduces new challenges concerning the trade-off between expressivity and scalability, especially in a streaming scenario [14].

Our approach is based on the assumption that not all raw data from the input stream might be relevant for complex reasoning, and efficient stream query processing can provide aggregation and filtering functionalities to help reducing the information load over the logic-based stream reasoner that can reason about conflicting, noisy and missing information [2]. Further investigation on reactive ASP reasoning [33], shifts the emphasis from rapid data processing towards complex reasoning and considers continuous and window-based reasoning over streaming data within the reasoned for better optimization. The resulting system OClingo [27], has been proven to outperform other non-monotonic reasoners [3], and more recent development around this approach are confirming the potentials of ASP-based stream reasoning [32], but its applicability to stream reasoning requires tighter coupling with efficient underlying stream processing techniques. For this reason, in CityPulse we will work on the interplay between stream query processing for stream filtering and aggregation, and stream reasoning with Answer Set Programming as a good starting point for investigating the applicability of complex reasoning over streaming data. This relates to the contextual filtering capabilities of the CityPulse adaptation component, which needs to process incoming semantic events and perform contextual reasoning to identify their relevance with the current ongoing task, but it also plays a crucial role for decision support capabilities, as will be detailed in future deliverable D5.2 due on M24.

The core task for the CF component and the algorithm used for assessing the contextual relevance and criticality of events will be detailed in Section 3.2 of this deliverable, while the implementation details based on ASP will be provided in Section 5.3.

3. Adaptive Urban Reasoning: From Semantic Streams to Dynamic Control

Within on-the-fly scalable processing from data to events to knowledge, the aim of Activity 5.1 (Real-Time Adaptive Urban Reasoning) is to i) identify which unexpected events among those identified by WP3 could affect the optimal result of the user-centric decision making support from Activity 5.2, and will provide adaptive feedback to the DS component by triggering new configurations that take into account the potential effects of unexpected events on the decision making process. In this way, results of the user-centric decision support can be continuously and dynamically updated in context-aware data stream processing in order to reflect not only the user's social media profile, situational awareness (e.g., location), preferences and application usage patterns as in Activity 5.2, but also on-going (and potentially unexpected) changes in the real-world environment and their effects on decision making.

The approach for Real-time adaptive urban reasoning is thus mainly based on the identification of adverse events that can potentially affect the decision support outcome. This is achieved by continuous query processing over semantic streams. Detected events will act as triggers for Adaptive Urban Reasoning to first assess the criticality of the detected events and then determines which adaptation is needed to ultimately provide better decision support. Notification of the adverse events is fed back to the appropriate component, which is responsible to perform the adaptation. Such adaptation can happen at the higher decision support level, or at the resource discovery and federation level via dynamic reconfiguration of the matchmaking process.

To summarize, activity 5.1 (Real-Time Adaptive Urban Reasoning) is composed of two main functional components for adaptability:

- ***Technical Adaptation component***: provides a higher-level control to configure event service stream federation, discovery and mash-up in WP3;
- ***Contextual Filtering component***: detects and filters unexpected events that might affect the optimal result of the user-centric DS (Activity 5.2).

The high level representation of these two components, their I/O flow and their interaction with other WPs is depicted in Figure 1 and will be detailed in the remainder of this section. Two different colours highlight the two separate control loops, namely TA and Contextual Filtering. The grey components are not relevant for this Deliverable but have been detailed in D2.2 and will be deployed and described in details in future deliverables within WP5.

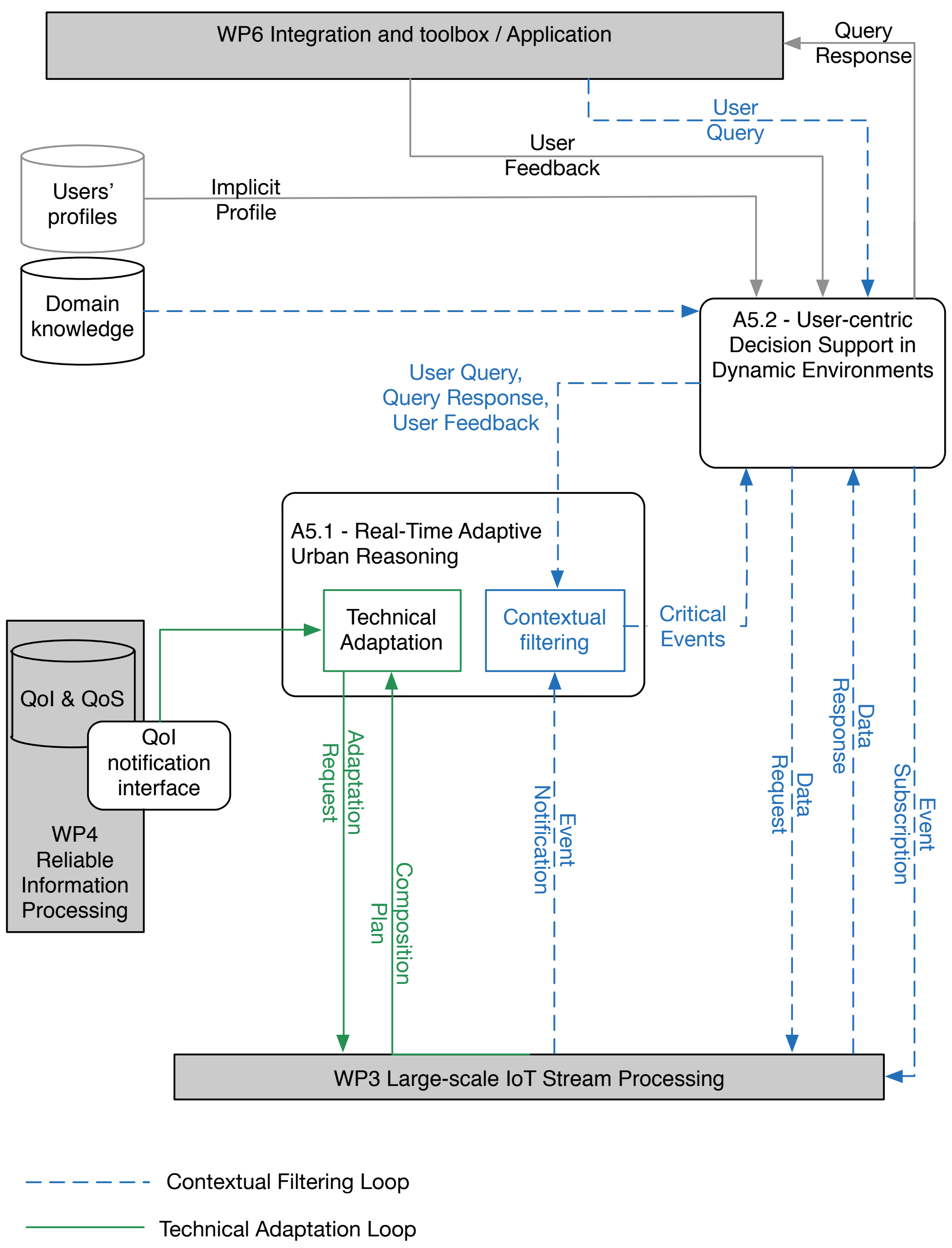

WP6 Integration and toolbox / Application
Query Response
User Query
User Feedback
Users' profiles
Implicit Profile
Domain knowledge
A5.2 - User-centric Decision Support in Dynamic Environments
User Query, Query Response, User Feedback
A5.1 - Real-Time Adaptive Urban Reasoning
Technical Adaptation
Contextual filtering
Critical Events
QoI & QoS
QoI notification interface
WP4 Reliable Information Processing
Adaptation Request
Composition Plan
Event Notification
Data Request
Data Response
Event Subscription
WP3 Large-scale IoT Stream Processing
Contextual Filtering Loop
Technical Adaptation Loop

Figure 1 – Real-time adaptive urban reasoning information flow

3.1 Loop 1: Technical Adaptation (TA)

This adaptation functionality is concerned with providing a higher-level control to configure data streaming in WP3. The configuration control relates to the ability to find the best (with respect to the NFPs) information sources to answer a particular query from the smart city application. Semantic query over distributed streams often involves the integration of different annotated streaming sources that are exposing the result of IoT stream processing as semantic event services (provided as a result of the activity 3.1 and 3.2 from WP3). The same information can be provided by different semantic event services, and the quality of such services is subjected to changes over time.

Therefore, this activity needs to cater not only for the discovery and selection of the best combination of such services (when the query is evaluated), but also for the ability to continuously detect whether the quality of information requirements are no longer matched. When this happens, the TA component should be able to adapt to changes by selecting *better* semantic event services at run-time, i.e. event services that can match the application requirements on the quality of information.

As an example, we consider a simplified version of the scenario n.1 “*Context-aware multimodal real-time travel planner”* documented in deliverable D2.1. A commuter uses a smart city application to keep track of the estimated travel time on his current route to work. In order to get the best possible route, the application needs to find the highest quality information all the time, possibly by integrating traffic information from multiple IoT data streams (different traffic sensors deployed on the road segments constituting the chosen route) as well as social streams (e.g. twitter feeds from users indicating some delays in portions of the path). The user may also specify a quality constraint on the estimated time, e.g.: the overall accuracy of the estimated time should be above 90% for any portion of the path). The technical work in WP3 is capable of selecting the IoT streams (among multiple candidate IoT devices/services) conforming to this accuracy requirement at design time based on static IoT stream description (e.g. by selecting traffic sensors with accuracy above a certain threshold). However, this capability cannot foresee or react upon the quality changes of the sensors at runtime, or trigger a new selection of more reliable sources when needed. For example, if the accuracy of a twitter feeds is directly proportional to the distance between the location of the tweet and the location the tweet refers to, whenever one of the chosen sensors reports an accuracy drop to 80% due to power loss, electro-magnetic interference etc., the accuracy of the federated IoT stream reporting the estimated travel time is compromised and the user's quality constraint is no longer satisfied. In this context, an adaptation strategy is required at run-time to deal with such technical issues by selecting, if available, traffic data provided by twitter feeds that are very close to the portion of the path where the delay occurs. In this way, the results provided by the application can rely on an underlying mechanism that maintains the quality levels of the federated IoT streams at any time.

Approach

The application requirements related to the NFPs are represented as constraints and preferences on QoI and QoS parameters[2]. Each semantic event service is associated to a list of QoI and QoS parameters that are maintained and updated by the activities in WP4 and exposed via a Notification Interface.

A query from the smart city application is represented as a complex event request that requires the generation of an event composition plan matching QoI and QoS requirements (this is performed by activity 3.2 in WP3: an aggregated quality capability is computed for candidate composition plans and matched against the quality constraint in the request, suitable candidates are then ranked based on the quality capability, constraint and preferences using a Simple-Additive-Weighting based calculation). Once this plan is generated, the TA component subscribes to updates of the QoI and

[2] A taxonomy of these parameters and their semantic representation is provided in Deliverable D3.1.

QoS information through the Notification Interface provided by WP4. When an update is detected for the QoI or QoS of one or more semantic event services, the TA component checks that the constraints and requirements specified by the application are still satisfied by the current composition plan. If this is not the case, an adaptation request is fed to WP3 requesting the generation of a new composition plan, and the current monitoring process is stopped and different subscriptions will be triggered based on the new composition plan provided by activity 3.2 in WP3. This makes sure that results returned to the application always meet the application requirements.

Unexpected Events

In the TA loop, unexpected events are caused by the changes in QoI/QoS for event streams producing data for a particular query/application. Such changes, in order to be identified as unexpected events and trigger an action consisting in an adaptation request, need to be significant to the point that they violate the original QoI/QoS constraints specified by the query/application.

In our approach to TA, we follow an *"if not broken don't fix"* strategy: a reaction is required only if the QoI/QoS requirements of the selected information sources drops. Maintaining an optimal federation at all times would require re-evaluating the composition plans not only upon quality improvements of all sources, but also upon any new event request sent by the application or any new stream registered in the system, in case the new stream fits better in the composition plan. While this is technically this is doable, it raises concerns about the overhead of switching back and forth over a very dynamic and large scale event service network, and might cause an unacceptable performance drop for event stream discovery and composition in this dynamic setting [26].

Management of changes in this case happens via subscription and via monitoring, and relies on the TA to have access to the description of the conflict evaluation mechanism used to check that a change is indeed an unexpected event.

The criticality of a quality change is assessed based on the latest quality information of all of the IoT streams involved in the composition plan. Whenever there is a quality update, the TA System retrieves the latest quality information of the updated stream and invokes the quality aggregation algorithm (specified during data federation) to calculate the aggregated quality vector for the composition plan. If the aggregated quality vector does not satisfy the quality constraint anymore, then the quality update is considered critical and sub sequential measures are taken to maintain the quality of the federated IoT stream.

Interactions with other WP components

The conceptual interactions between the TA component and the other functional components in related technical work packages is illustrated in Figure 2 (please refer to Figure 1 for an overall view of the TA process flow).

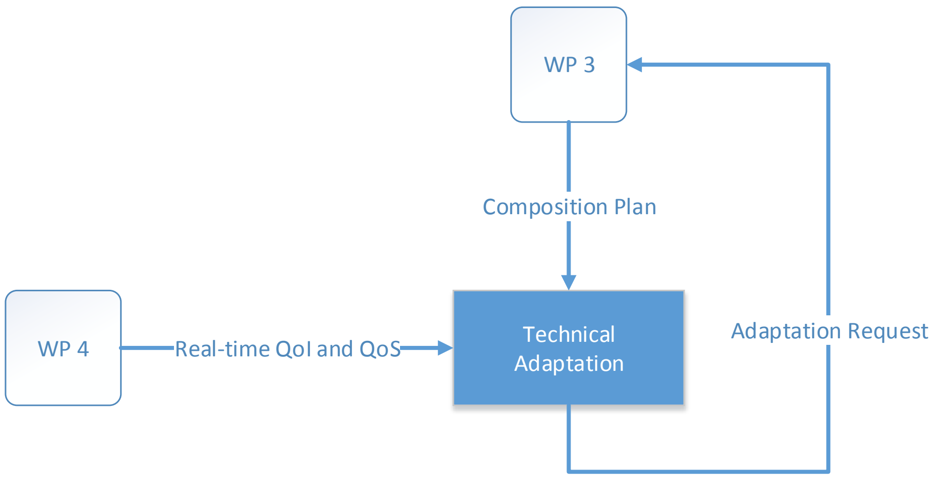


Figure 2 – Technical adaptation I/O

The TA component subscribes to real-time QoI/QoS changes continuously monitored and updated by WP4. As part of the input for TA, the composition plan determined for each query/application request is also provided by WP3.

The output of TA is an adaptation request provided to WP3 in order to trigger the generation of a new composition plan as a result of a change in QoI/QoS that violates the initial quality constraints in the event requests.

3.2 Loop 2: Contextual Filtering

The Contextual Filter aims to provide adaptive feedback to the DS system by triggering new configurations that take into account the potential effects of unexpected events on the decision making process. In this way, we can present instantaneously the new results based on the changes of the environment. The CF component is able to understand if an on-going change in the real-world (an event) can affect results provided to the user, making them invalid or not useful anymore. Moreover it helps in the “dynamic” queries because it can detect if there are updates of the “events of interest”.

Consider the travel planner example defined in Section 3.1: a user wants to find a path to travel from a starting point ***S*** to a goal point ***G*** minimizing the time. The DS component should not only be aimed at providing the most suitable answer when the application is launched, but it should also provide continuous support for selecting the best possible route when there are some critical events which can potentially effect the commuter while being on-the-move. This is in principle what happens with the re-routing functionality of modern satnav systems when the user changes the path, except that off-the-shelf systems do not rely on interoperable solution to cater for unexpected events and adapt to select the best possible data sources available at any time. In order to realize the capability of reacting to unexpected events, the DS component subscribes to a list of events that are relevant for a particular application and for a particular reasoning task (which are specified at design time), e.g. accidents, road traffic, flooding, road diversions and so on. The occurrence of any of such relevant unexpected events is notified to the CF Component by WP3, and metadata describing some details of the events is also provided. With this information, the CF component is in charge of selecting among the list of detected unexpected events, the ones that are really relevant according to the context. In our travel planner example, changes in traffic congestion level may affect the actual path of the user if they will be solved in reasonable time and the user is still far away, or road diversions will not be a major deal if the location of the user along the path has already gone through the area where the diversion occurred. The level of criticality of events is also dynamically assessed by the CF component based on some metrics that will be detailed in the remainder of this section. Based on this level of criticality of related events, the DS component will take the appropriate course of action, e.g. re-compute the alternative paths for the user based on the new conditions, or simply inform the user about the critical event and its possible effect and let the user provide a feedback.

Approach

Once a reasoning process has been triggered, the user query/application request obtains an answer through the DS component (these aspects will be detailed in the future deliverable D5.2). As mentioned earlier in this section, the DS component also subscribes to a list of high-level events produced by WP3 that are relevant to that particular application or request.

A graph that relates those events with the properties describing the user context is generated based the Definitions 1-6 introduced later in this section, and according to the contextual elements related to a DS module (which will be detailed in future deliverable D5.2). Whenever any of the potentially relevant events is detected, the CF component is in charge of determining what is the impact of such an event for the DS outcome. In the current (first instance) of the component, such relevance or level of criticality is computed as a cost metric based on the distance of the event and the user context properties. The use of application-specific thresholds is in scope here to filter out events that are "not enough" critical, but we will introduce it in future releases of the component.

When a critical event is detected, the CF component notifies the DS component with some metadata about the possible required adaptations. The DS component will then decide whether the computation of new solutions is needed with new requirements that can reduce the impact of the critical event.

For example, if an accident is detected as a potentially critical event for the travel planner DS module, the CF component will relate such event with the position of the user along the path, and to the expected time of co-occurrence of the user position in the area of the accident (in case the information about the duration of such even is available). As a result of this correlation analysis, the CF component will be able to find out whether the accident is far from the user position and is likely to be solved by the time the user gets there, or whether a new path needs to be computed by the DS component with the requirement to avoid the area where the accident occurred or where the traffic is congested because of the accident. The DS component is then in charge of the formulation of such requirements as constraints and will cater for the computation of new solutions.

Unexpected Events

In the CF adaptive functionality, unexpected events are conceptually defined as those events that will more likely impact on the effectiveness of the reasoning outcome (either making it invalid or not compliant to the user preferences and requirements). In a way, the intuition is similar to how unexpected events are described and managed in the TA loop, but with the complication that here we are operating on a higher level of abstraction, therefore the verification of the criticality with respect to a reasoning task is not as straightforward.

First of all there is the concept of an event being “relevant” for a reasoning module or task. In the first instance of the implementation of the smart city framework, relevant events are associated to reasoning tasks through the Event Ontology defined at design time.

When new events are added to this ontology, the list of events to be considered in the CF process needs to be updated. The incremental changes are not included in the current version; however the use of a flexible and extensible ontology for describing the events makes for an easy inclusion of future new events into the ontology as they are introduced. Once a new event is included in the ontology there should be relevant processing and interpretation methods that are responsible for detecting and processing that event. For further technical details we refer to the deliverable D3.4 (will be released in M30).

Here are some definitions that will help us understanding what a critical event is and some principles for the design of automatic filtering algorithm.

Definition 1: An event *graph* G consists of two components: G = (E, R) where

- E is a set of nodes divided into two disjoint subsets
    - EU is a set of unexpected events that are detected by WP3 (activity A3.4)
    - EC is a set of contextual elements that are related to contextual information and can depend on some user preferences and requirements
- R is a set of edges in the graph representing relations between an element in EU and an element in EC. In other words the edge says that an unexpected event in EU affects a reasoning module depending on a contextual element in EC.

Definition 2: Each edge representing a relation in R has a weight, and the weight of an edge from eu_i to ec_j is represented as $w(eu_i, ec_h)$. The weight says how critical eu_i is for the reasoning task based on the contextual element ec_h.

Definition 3: Given a contextual element *ec*, a partition function **F** over **EU** with respect to *ec* is defined as

F : **EU** → $\{EU_k, k=1..n\}$

F generates n partitions of EU based on admissible values of the contextual element *ec*. We use these possible values as labels l_k for each partition EU_k and we impose an order between such partitions such that

$EU_i \leq EU_j$ if the impact of events in EU_j on *ec* is greater than the impact of events in EU_j on *ec* when $ec=l_k$. With this ordering, we assign weights to an edge as per Definition 2 such that for each i,j with $EU_i \leq EU_j$ we have that $w(eu_i, ec) \leq w(eu_i, ec)$ with eu_i, in EU_i and eu_i in EU_i.

This operation is repeated for each contextual element ec_h in *EC,* resulting in several partitions of the same unexpected events in relation to each contextual element.

Definition 4: For each unexpected event *eu* in EU, we compute the total criticality measure for *eu* with respect to all of the contextual elements ec_h in EC, such that there is an edge between *eu* and ec_h, defined as follows

$$w(eu) = \Sigma\, w(eu, ec_h) \qquad s.t.\ eu \text{ is in EU and } ec_h \text{ is in EC}$$

Once this weight is being computed, it gives us an order over the events indicating which event is more critical than the other. A naive approach would now be that of considering only the unexpected events with highest criticality at each interaction. A refined version would be that of using an application-specific threshold α and say that an unexpected event *eu* is critical if $w(eu) \geq \alpha$.

Definition 5: Based on Definitions 3 and 4, we can define the impact of a particular contextual element *ec* as the sum of the weights of all unexpected events in each partition EU_k in EU as

$$w(ec) = \Sigma\, w(eu_j, ec) \qquad \text{s.t.} \qquad eu_i \text{ is in EU and } ec \text{ is in EC}$$

Note that for each pair i, j we have that $EU_i \cap EU_j = \varnothing$ (all partitions are disjoint). The value of $w(EU_k)$ can be used to make further analysis on the importance of different contextual elements for a particular reasoning module, as it will be detailed in the DS deliverable D5.2.

Example: A user is traveling from a starting point ***S*** to a goal point ***G*** by car minimizing the time needed to get to ***G***. His initial path provided as answer from the DS component is going from ***S*** to ***G*** through two mid points ***A*** and ***B***, and the current location is the starting point ***S***. The position of the user is a contextual element ec_1*=position(S).* The DS component subscribes to the occurrence of unexpected events that are relevant for the specific DS module (in this case events on the path from S to G) from an event list created from the Event Ontology, and detects the following unexpected events:

- an accident eu_1*=accident(S,A)* in the path between S and A
- a traffic jam eu_2*=traffic(A,B)* in the path between A and B
- a flood eu_3*=flood(B,G)* in the path between B and G

An event graph is created for the contextual element ec_1 and the unexpected events eu_i, i={1,2,3}, with edges directed from each eu_i to ec_1. In order to compute the weights on such directed edges, we consider a partition function that classifies each event eu_i based on their location with respect to the partitions in the path. Thus we divide the space into three sets corresponding to the three segments of the path labeled SA, AB and BG (see Figure 3) such that

- SA = {eu_1};
- AB = {eu_2}
- BG = {eu_3}

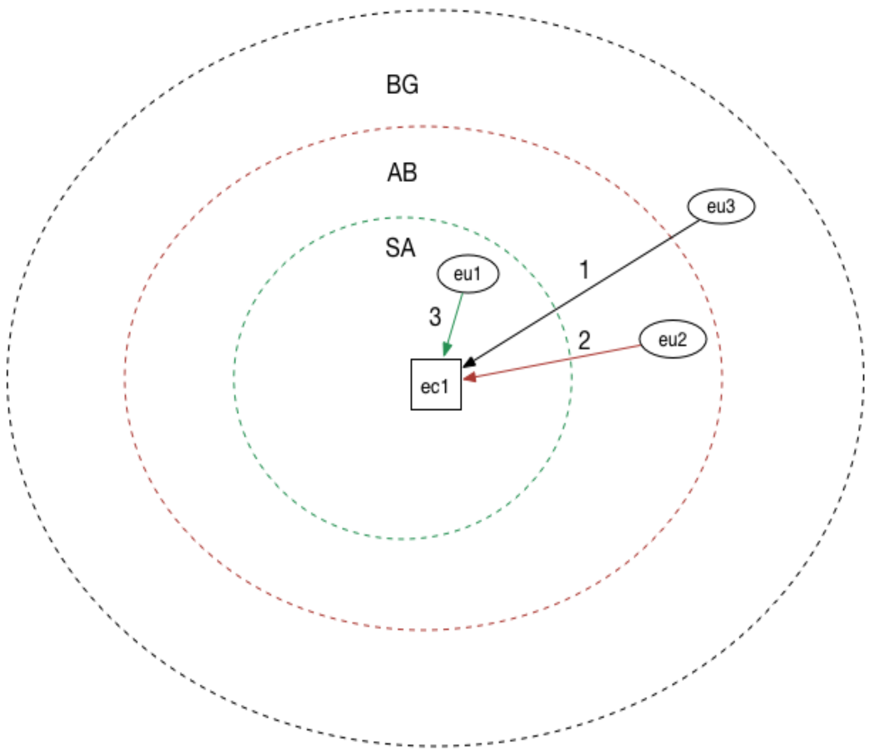


Figure 3 – Partition related to the position of the user as contextual element

Based on this representation let's assume the user is in the segment SA, the order between the partition is BG ≤ AB ≤ SA. We reflect this in the weights assigning minimum weight equal to 1 to unexpected events in BG, and increasing the weight by 1 for AB and by 2 for SA respectively. This would satisfy the order imposed by Definition 3 and results in the graphical representation in Figure 3.

Now let's consider the contextual element *ec_2 = transportation_type(T).* Since the labels can be either "bike" or "car", we say that we have two possible orders:

- bike ≥ car if the user is going by bike
- car ≥ bike if the user is going by car

Let's assume in this example that the user is going by bike, weights are assigned to unexpected events in the two partitions accordingly so that the order imposed by Definition 3 holds. The graph and weights are depicted in Figure 4.

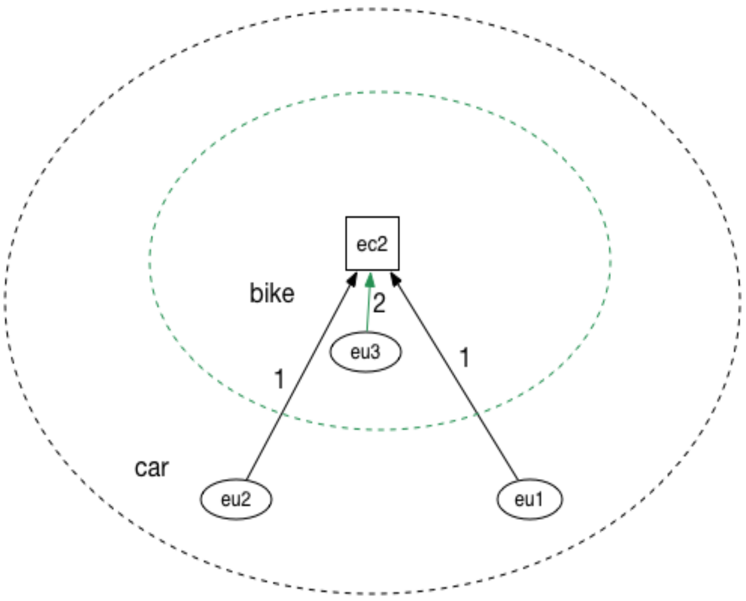


Figure 4 – Partition related to the type of transportation as contextual element

For each unexpected event, if we calculate the total criticality measure as per Definition 4 we obtain:

- $w(eu_1) = w(eu_1,ec_1) + w(eu_1,ec_2) = 3+1 = 4$
- $w(eu_2) = w(eu_2,ec_1) + w(eu_2,ec_2) = 2+1 = 3$
- $w(eu_3) = w(eu_3,ec_1) + w(eu_3,ec_2) = 1+2 = 3$

Furthermore, the impact of each contextual element as per Definition 5 is as follows:

- $w(ec_1) = w(eu_1,ec_1) + w(eu_2,ec_1) + w(eu_3,ec_1) = 6$
- $w(ec_2) = w(eu_1,ec_2) + w(eu_2,ec_2) + w(eu_3,ec_2) = 4$

which indicates intuitively that contextual element ec_1 is more relevant than ec_2, although this analysis should be performed over several cycles of unexpected events and related partitions to obtain a more realistic estimate.

With this characterization, the event that will be considered as critical in this cycle of the CF would only be eu_1. The use of threshold α could also be used to determine which event is critical with respect to the reasoning task. An initial approximation of α could be that of considering the average weight of all unexpected events, approximated to the immediately higher natural number. In this case, we would have that α=4, so that only eu_1 would be considered as critical, but more sophisticated ways of defining this threshold will be considered in future revisions.

Whenever an unexpected event is labelled as critical by the CF based on the computation of such level of criticality, a notification will be sent to the DS component, which will take those events into account when computing the new answer. For details on how the notification is managed and which actions are triggered by that will be documented in the future deliverable D5.2 (will be released in M24)

Interactions with other WP components

The conceptual interactions between the CF component and the other functional components in related technical work packages is illustrated in Figure 5.

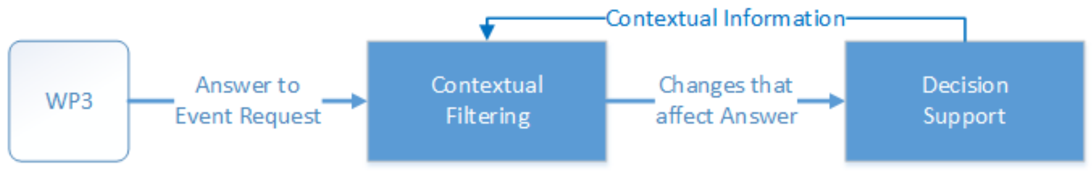


Figure 5 – Contextual Filtering I/O

The CF component gets i) an answer to event requests returned from WP3 (list of events) and ii) Contextual Information from the DS component (including user query, query response, and user feedback when available).

The output provided by the CF component consists of a list of critical events that affect query response, which is passed on to the DS component.

4. User-centric Approach

In this section we want to highlight aspects that make the adaptive reasoning user-centric. Some of these aspects will be more exploited in the DS component for user-centric solution (Activity A5.2, documented in Deliverable D5.2).

Despite some details of this focus on the user will be more relevant for DS in Activity A5.2 than for Adaptive Urban Reasoning, it is worth mentioning the intuitions that prove how they also play a role in the adaptation component.

4.1 User-centric Technical Adaptation: Preferences and Requirements

In this section we discuss how users' preferences and requirements play a role in the TA loops and how they are used to detect violations of user- and application-dependent constraints guiding the discovery and composition process

During the discovery and composition process, numerical quality vectors can be used to specify the qualities and constraints over the quality metrics for data streams. For example, if we consider some typical QoS attributes, including: latency, price, energy consumption, bandwidth consumption, availability, completeness, accuracy and security, a numerical quality vector Q=<L,P,E,B,Ava,C,Acc,S> can be used to specify the quality of a data stream w.r.t. these dimensions along within 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 within 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=<L_w,P_w,E_w,B_w,Ava_w,C_w,Acc_w,S_w> 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, because it best suits the users' or applications' constraints and preferences. 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.

4.2 Contextual Filtering: profiles and feedback

Users' profiles store the description of the characteristics of persons and preferences. These characteristics and preferences are used as contextual information for the application. For example, in a travel recommendation application the ability to drive a car or the health condition of a user can influence the DS module about which transport to use –car, bicycle, bus, etc.-. In the case of an unexpected event, some assumptions could be invalid and the user's profile could again influence the decision module. Therefore, users' profiles and preferences play a role in CF to determine the user context and how this context affects events' relevance.

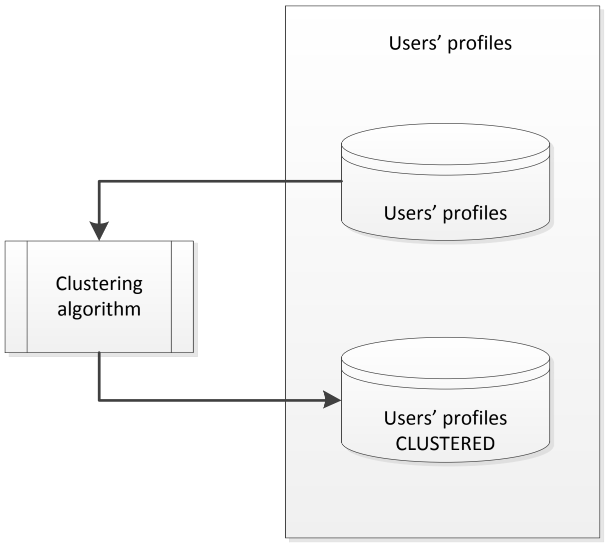


Figure 6 – Users' profile management

A user profile is included in a knowledge base every time a new user accesses the CityPulse framework through one of the Smart City applications relying on it. The contextual filtering uses this profile to adapt the DS system accordingly. However, sometimes the profiles are not complete and the CF could not get accurate information about a particular user. In these situations, some techniques have been used to group users in different clusters according to their profile and assume that the missing information about a particular user should be equal to those users with similar profiles. These techniques are unsupervised learning algorithms, which are able to group users based on a set of variables. The clusters are stored in a knowledge base (see Figure 6). If a new user only fills in some variables, the user profile system could derive the most suitable cluster for that user and infer the missing information with that of the cluster. For example, a new user fills in a few variables (e.g. low income, student, young), but he does not fill in his preferred transportation medium. The system compares the new user profile with existing users and assigns a group to that user. Within that group the most common transportation medium is the bus. Therefore the user profile system could guess that the new user would prefer to travel by bus. The user profile, with the inferred variables, is sent to the CF for further processing.

The user profile module calculates the clusters only when a relevant number of users have joined the platform, because this recalculation involves the execution of learning algorithms over a huge amount of data, and it could be time consuming. Therefore, every time a new user access the system, the user profile module only measures the distance between said new user an each cluster, which involves little processing time.

The user profile knowledge base is updated every time a new user accesses the platform, but also it is updated with the user feedback (aka. the selections the user makes in the application).

As for privacy protection, users' profiles will be aggregated into classes (or clusters) and will not contain exact values of profiles attributes that might identify a particular user. For example, ranges of values for age would identify users with age between 30 and 50. Likewise, within the users' profiles knowledge base only the users of a specific application should have their profile accessible by that application, whereas the whole clustered users' profiles (aggregated) should be accessible by any application. This clustered knowledge base will contain only clusters which depend in several variables, such as young people, student, low or medium income, able to drive, etc. This clustered information not only anonymises the information, but it also makes difficult the inference of prejudged statements related to gender, age, religion or other sensitive information.

5. Implementation Details

In this section we provide some implementation details of the current functionalities of Real-Time Adaptive Urban Reasoning. A simplified (for readability reasons) UML representation of the physical deployment of the components is presented in Figure 7.

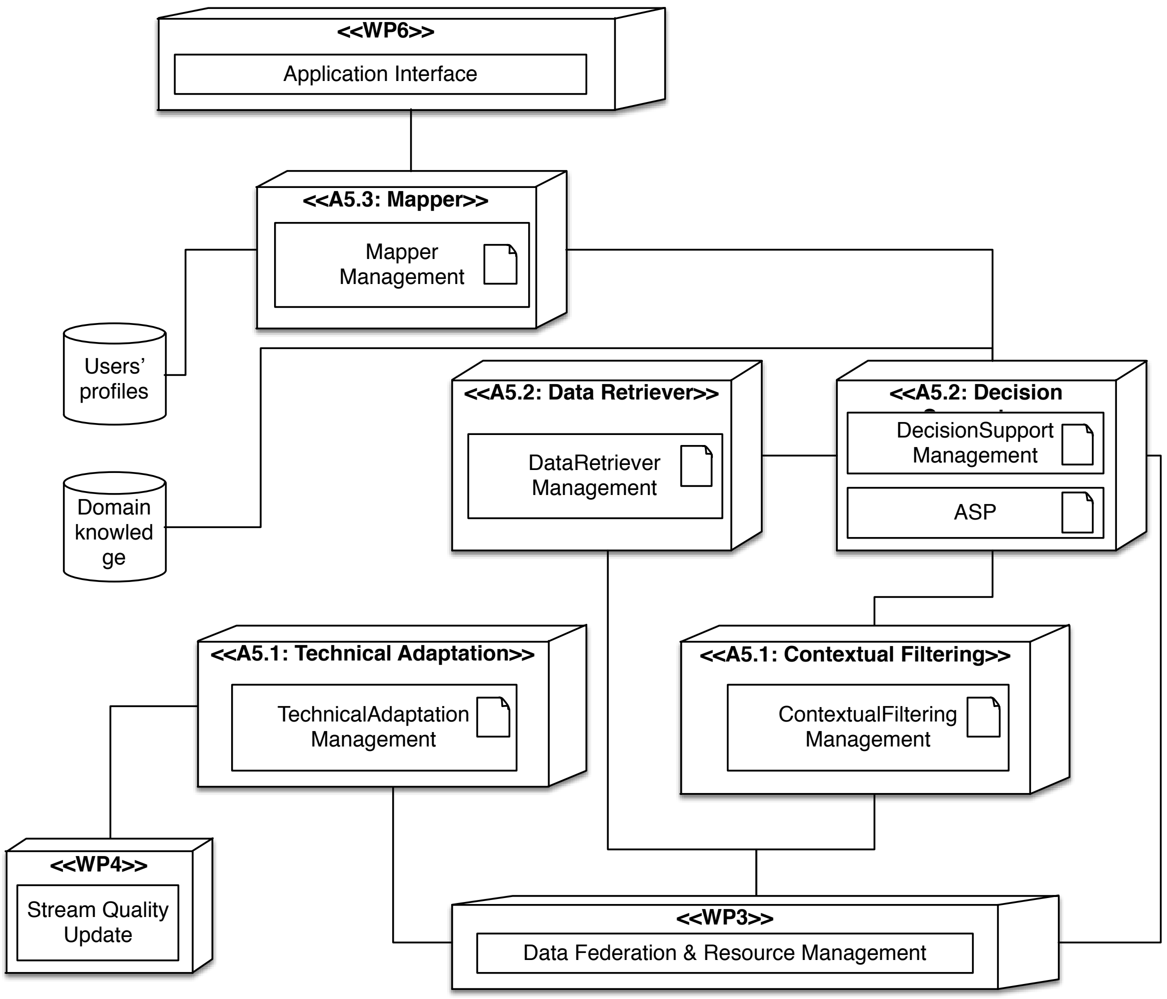


Figure 7 – Deployment Diagram for Real-Time Adaptive Urban Reasoning

As detailed in the previous subsections, the adaptation functionality of WP5 is composed by i) a TA to adapt discovery and composition of Event Streams based on QoI and QoS parameters, and ii) a CF component which operates at the level of unexpected semantically annotated events and determines their criticality for the DS process. In what follows, we firstly provide a detailed description of the underlying technologies and tools used by both TA and CF, in the later part of this section, we present separate sequence diagrams for each of the two processes and provide a description of the main implemented modules.

5.1 Related Technologies

TA and CF components utilise some of the existing tools to perform their relevant tasks. We briefly discuss each of these tools and also elaborate how these tools communicate with our components.

Linked Stream Middleware: Linked Stream Middleware (LSM) makes it easy to integrate time-dependent data with other Linked Data sources, by enriching both sensor sources and sensor data streams with semantic descriptions, and enabling complex SPARQL-like queries across both dataset types through a novel query processing engine, along with means to mashup the data and process results. Most prominently, LSM provides (1) extensible means for real-time data collection and publishing using a cloud-based infrastructure, (2) a Web interface for data annotation and visualisation, and (3) a SPARQL endpoint for querying unified Linked Stream Data and Linked Data [25].

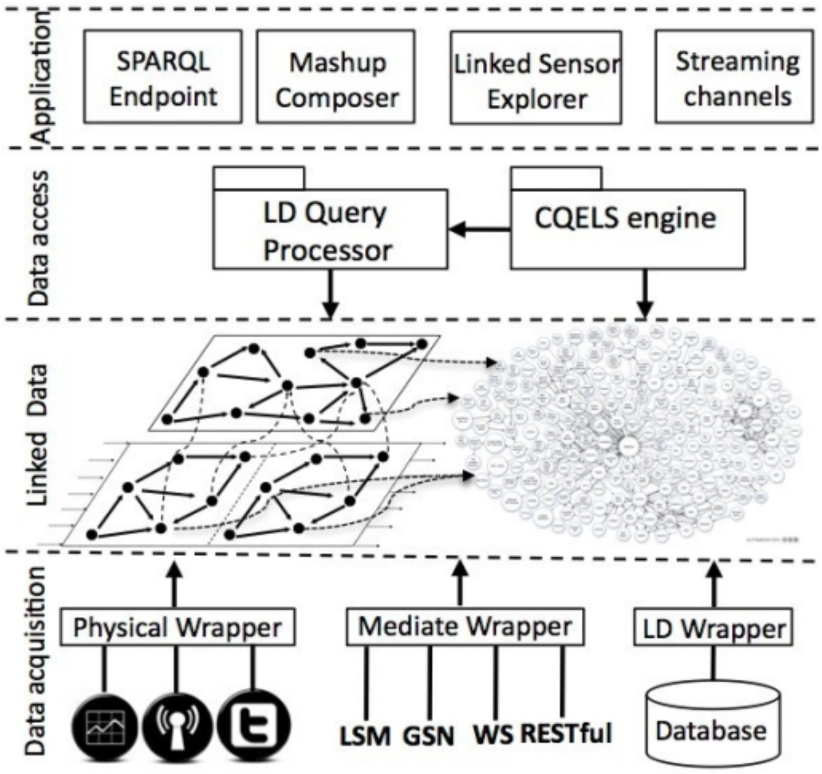


Figure 8 – LSM Layered Architecture

The LSM architecture, as shown in Figure 8, is divided into four layers. The Data Acquisition Layer provides three wrapper types: Physical Wrappers that are designed for collecting sensor data from physical devices; Linked Data (LD) Wrappers that expose relational database sensor data into RDF; and Mediate Wrappers, which allow for collection of data from other sensor middleware such as Global Sensor Networks (GSN). However, our Smart City Framework is not limited to these wrappers and can be extended to include additional wrappers for different data formats. The Linked Data Layer allows access to the Linked Sensor Data created by the wrappers and can also link it with the static datasets of linked data e.g. Linked Open Data (LOD). The Data Access Layer provides two query processors, a Linked Data query processor and the Continuous Query Evaluation over Linked Streams (CQELS) engine, and exposes the data for end-users or machine-users. The fourth layer, the Application Layer, offers a SPARQL (SPARQL Protocol and RDF Query Language) endpoint, a mash-up composer, linked sensor explorer, and streaming channels.

Continuous Query Evaluation over Linked Streams: CQELS is a native and adaptive query processor for unified query processing over Linked Stream Data and Linked Data. In contrast to the existing systems, CQELS uses a "white box" approach and implements the required query operators natively to avoid the overhead and limitations of closed system regimes. CQELS provides a flexible query execution framework with the query processor dynamically adapting to the changes in the input data [24].

Standard query processing techniques for Linked Data cannot be directly applied for processing Linked Stream Data, due to its dynamic nature: while traditional Linked Data queries are executed once over the entire collection and discarded after the results are produced, queries over Linked Stream Data are continuous. Continuous queries are first registered in the system, and then continuously executed as new data arrives, with new results being output as soon as they are produced.

For processing continuous queries over Linked Stream Data, the LSM provides the CQELS engine. The query processing in CQELS is done in a push-based fashion, i.e., data entering the query engine triggers the processing. As soon as any result is found, it is sent to the query listeners, which are used to push the result output data stream of a continuous query to the delivery channels (result stream). The continuous queries are expressed in the CQELS language, which is an extension of SPARQL 1.1.

CQELS is an integral part of the LSM framework and can be considered as the core of the LSM. CQELS engine executes the queries registered over LSM and can execute those queries over live data streams. CQELS has also capability to perform live queries over the combination of the data arising from live streams as well from static data sources.

Clingo4: State-of-the-art ASP solvers all follows, internally, a two-step approach better known as generate-and-test. First, a grounder generates a (finite) propositional representation of the input program; then a solver computes the stable models of the propositional program by exploring the search space to find solutions that are entailed by the program rules (test). This control loop is pre-defined and the search is guided by heuristics that allow little user control.

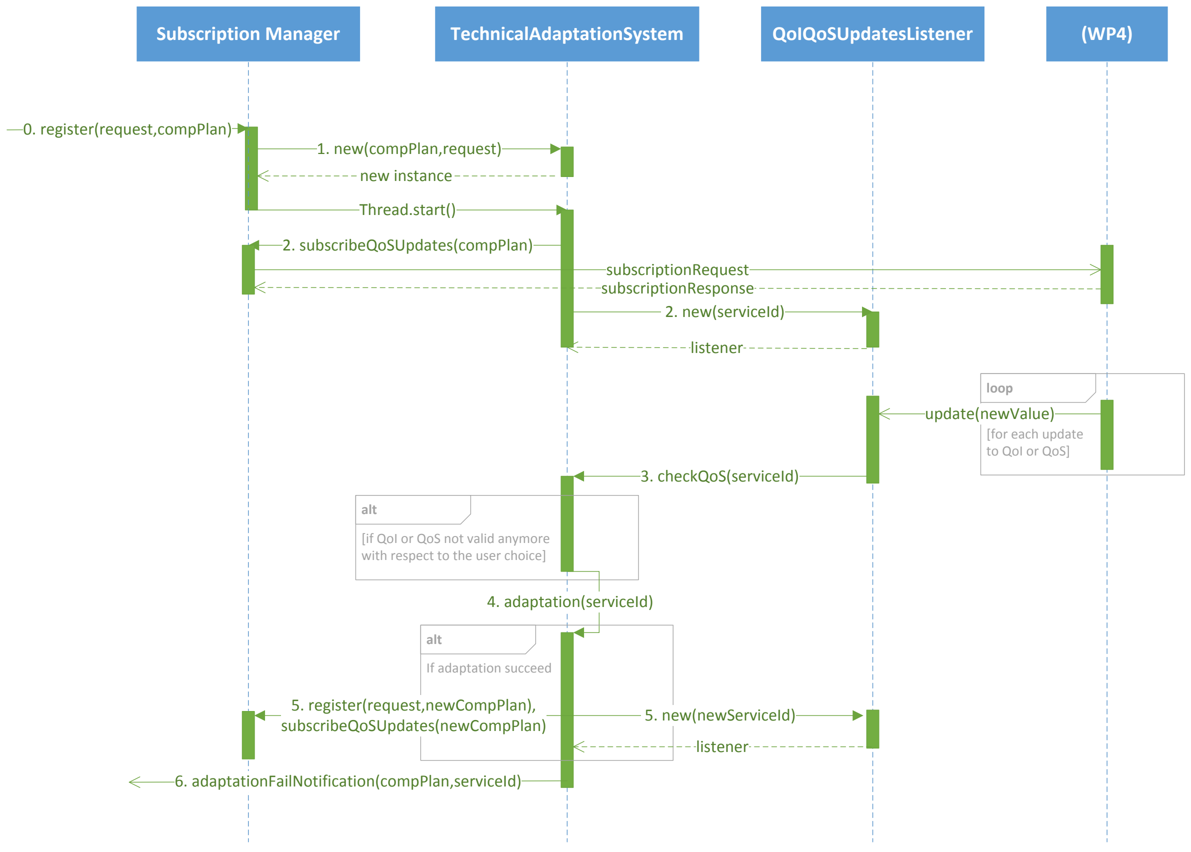
Clingo4 [32] is an ASP system that combines the grounder *gringo* with the solver clasp for computing stable models of ASP programs. The new clingo4 series provides new high-level constructs to deal with changes in the problem specification during the reasoning process (for example when data or rules are added, deleted or replaced). The grounding and solving steps in clingo4 are performed within a single integrated process to avoid redundancies in grounder and solver calls. On the declarative side, a flexible way to parameterise programs is also provided along with support for the ASP declarative input language and control capacities via embedded scripting languages Lua and Python. The separation of logic and control programs in clingo4 eliminates the need for special-purpose systems for incremental and reactive reasoning like iclingo [20] and oclingo [33].

5.2 Technical Adaptation

The TA component is responsible to continuously monitor the QoI/QoS scores of the all the contributing data streams relevant to a specific event request. TA component comprises of three main modules, namely:

- Discovery: This module receives a composition plan with a reference to the event request for which the composition plan was generated. Event request contains user/application functional requirements as well as non-functional requirements (QoI/QoS constraints and preferences). Discovery module discovers the relevant QoI/QoS update streams to subscribe.
- Monitoring: This module generates a subscription request for any update in the QoI/QoS scores of the relevant data streams. Monitoring module continuously monitors and verifies that whether QoI/QoS scores of all the contributing data streams in a composition plan are compliant to the event request.
- Adaptation Request: This module is triggered if any of the user defined non-functional constraints and preferences are violated, this module requests the federation component to re-discover the data streams, which qualify to the user constraints and preference, described in the original event request. Federation component will send the new composition plan to the technical adaptation, which will resume discovery and monitoring module for the new composition plan. However, if the Federation component is unable to re-generate new composition plan because none of the available data stream comply with the constraints and preferences defined in the event request. The TA component provides a feedback to the user/application to reconsider the constraints/preferences and generate a new event request.

Figure 9 depicts system sequence diagram to showcase the interactions of the technical adaptation modules with each other as well as with the external components.



Subscription Manager
TechnicalAdaptationSystem
QoIQoSUpdatesListener
(WP4)
0. register(request,compPlan)
1. new(compPlan,request)
new instance
Thread.start()
2. subscribeQoSUpdates(compPlan)
subscriptionRequest
subscriptionResponse
2. new(serviceId)
listener
loop
update(newValue)
[for each update to QoI or QoS]
3. checkQoS(serviceId)
alt
[if QoI or QoS not valid anymore with respect to the user choice]
4. adaptation(serviceId)
alt
If adaptation succeed
5. register(request,newCompPlan), subscribeQoSUpdates(newCompPlan)
5. new(newServiceId)
listener
6. adaptationFailNotification(compPlan,serviceId)

Figure 9 – Sequence Diagram for the Technical Adaptation loop

A detailed step-by-step description is as follows:

1. When a new *EventRequest* to detect complex events required by a user/application is registered at the Subscription Manager, an instance of the *TechnicalAdaptationSystem* is instantiated as a new *Thread*, and its *start()* function is called.
2. The newly instantiated *TechnicalAdaptationSystem* receives two parameters: *currentCompositionPlan* and a reference to the *eventRequest* for which the current CompositonPlan was generated. Discovery modules discover relevant QoI/QoS update streams for subscription. Once discovered it initiates the *subscribeQoSUpdate()* function in the *SubscriptionManager* to subscribe to those QoS update streams. Meanwhile, *QoSUpdateListeners* are instantiated to capture the QoS updates.
3. Whenever there is an update in QoI/QoS score, the listener invokes the *checkQoS()* function in the *TechnicalAdapatationSystem* to check if the overall QoI/QoS performance still conforms with the user/application constraint in the event request.
4. Whenever the QoI/QoS constraints are violated. The *TechnicalAdaptationSystem* invokes the *adaptation()* process with a parameter *serviceID* indicating which service performance is violating the QoS constraints. The *adaptation()* process tries to create a new composition plan conforming to the user's original QoS constraints.
5. If the adaptation succeeds, the *TechnicalAdaptationSystem* sends the new composition plan to the *SubscriptionManager*, who renews the relevant streams (regular event streams and

QoS update streams) to subscribe to. Meanwhile, the *TechnicalAdaptationSystem* updates its QoS constraint queries and listeners.

If the adaptation process fails, the *TechnicalAdaptationSystem* sends a notification to the user/application. The notification message contains which event request is affected, and caused by which member event service.

The TA component uses LSM to register monitoring queries over the streams used in the composition plan generated by the WP3. Composition plan contains complete details of the data streams used and their QoI/QoS requirements and preferences. TA component identifies relevant data streams for any composition plans and subscribes to the updates of those streams using LSM framework. Monitoring queries compare the QoI/QoS constraints defined in the composition plan with the latest QoI/QoS score updates provided by WP 4 and trigger appropriate actions if any of the constraint is violated. It is worth mentioning that another instance of the LSM and CQELS is an essential part of the Data Federation component of WP3. In order to avoid multiple instances of LSM and CQELS running within the Smart City Framework, WP 5 components adapted according the event request format developed as part of WP3. This adaption leads to better performance results. Details of the event requests and Data Federation and Optimisation will follow in subsequent deliverables (D 3.2).

5.3 Contextual Filtering

The CF component aims to provide the critical events, which can affect the answer, to the DS. This component relies on both the list of events detected from WP3 and contextual information sent from DS in order to filter the critical events. At the first time the user submits a query, this query will be considered as contextual information and will be sent to CF. However, as soon as the initial answer is provided to the user, CF will take this answer and user's feedback into account when it filters the critical events. These critical events are computed based on an event graph, which is created from the list of events and contextual information. The CF component comprises of three main modules, namely

- i) Discovery: This module receives events (both primitive and composite events) from WP3 and contextual information from DS.
- ii) GraphCreater: This module generates an event graph based on events and contextual filtering.
- iii) Filtering: This module filters the critical events based on a threshold. For each unexpected event detected from WP3, this module computes the weight of the path from that event to the contextual elements. The critical events are events that have weights are smaller then the threshold.

Figure 10 illustrates the flow of CF process. As illustrated in the figure, CF relies on the contextual information from the DS component and events that are detected from Data Federation in WP3 in order to filter the critical events. This contextual information is derived from user's query, query response, and user's feedback. Based on the events and the contextual information, critical events are filtered out. Finally, the filtered critical events will be sent to DS.

1. CF starts by calling getContextualInformation() function in order to get ContextualInformation from DS. This Contextual Information consists of user's query, query response, and user feedback.
2. At the same time, CF also calls the getEvents() function to WP3 to collect detected events.
3. Based on ContextualInformation and Events, CF invokes the *filterCriticalEvent()* process. This process tries to filter critical events out.
4. These critical events will be sent to DS for computing an update answer to the user.

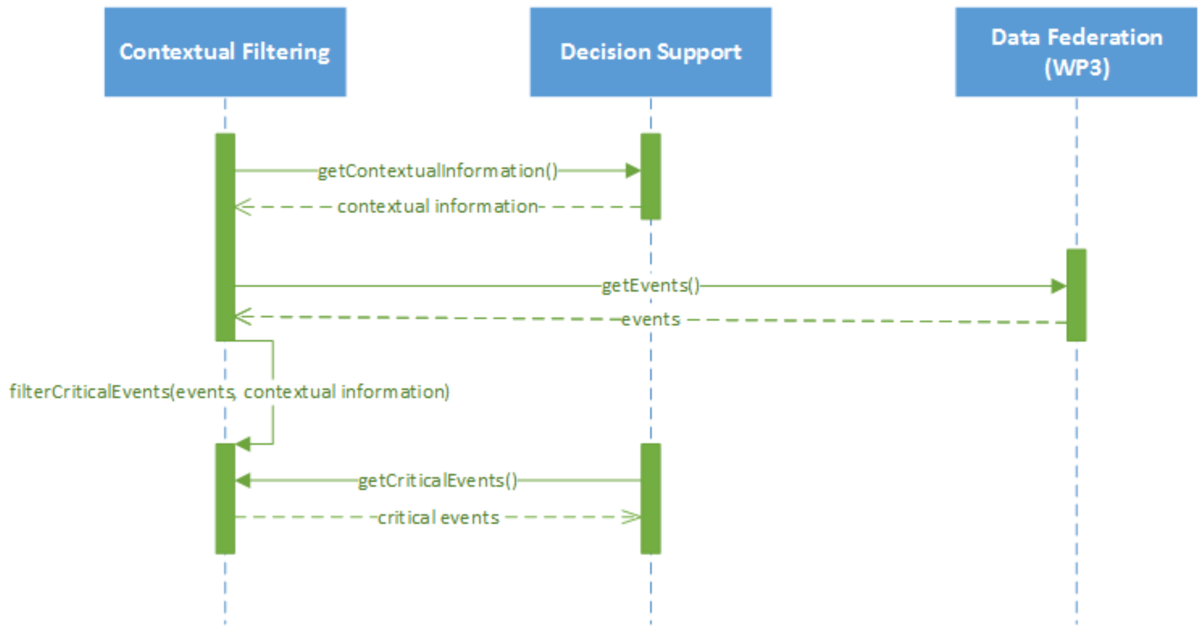


Figure 10 – Sequence Diagram for the Contextual Filtering Loop

The CF component requires to use the CQELS engine to execute the monitoring queries over live streams. A typical example of monitoring query for contextual filtering is getting current context of a user; e.g. location of the objects on the move. Similar to the TA component, the CF component also adapted to the event request format used in WP3 to avoid multiple instance of CQELS engine and achieve better performance within the Smart City Framework (details of how the CQELS instance is used in WP3 will be provided in future deliverable D3.2 due M24).

The implementation of the algorithm described in Section 3.2 relies on the *filterCriticalEvents()* method that takes in input the list of unexpected events provided by WP3 and the contextual information that are relevant for the particular reasoning task, and produces a set of events that are contextually critical and therefore require adaptation.

This method currently relies on an external call to the ASP solver Clingo to build and label the event graph, and on parsing mechanisms to transform Java objects into ASP facts. This choice introduces a little overhead in data transformation, but enables a direct consumption of the results of ASP reasoning for contextual filtering by the DS component that will perform complex reasoning calling the same ASP solver.

ASP results provided as set of facts would need to be then encapsulated in a structured way so that they can be consumed and exposed to WP6, but these steps will be part of the DS component and will be detailed in future deliverable D5.2 due M24).

In our first implementation we consider location and time as main dynamic contextual aspects that relate unexpected events and dynamic user context. We also consider dynamic contextual information provided by the DS component and associated to the user status (such as user's location) and the user preferences (such as user's transportation type).

Changes in the real world detected by WP3 as unexpected events are transformed via script into ASP facts as predicates (note that upper case parameters denote variables while lower case parameters denote constants) of the form

```
detected(EventID, Level, SegmentID, T).
```

where

- `EventID` is the ID of an unexpected event
- `Level` is the quality of an unexpected event
- `SegmentID` denotes where the event happens (in OSM segments)
- `T` is the time when the event was detected

For the moment we use randomly generated IDs for the event, and we use a discrete relative time stamp represented as a natural number. This will be changed based on how events will be produced by activity A3.4 and will not require changes in the implementation, as long as an ordering relation is computable between two different timestamps.

Contextual information is provided by the DS component as a set of ASP rules describing influence relations between unexpected events (expressed as changes in the environment) and contextual aspects (both dynamic and static such as the user profile or preferences). Details about these rules and the model used to represent such contextual dependencies will be provided as part of the future deliverable describing the DS component, D5.2, due on M24.

Since events from WP3 are produced asynchronously, the ASP rules implementing the algorithm in Section 3.2 in clingo4 (combined with rules for contextual dependencies from DS) will cater for the inclusion of undefined atoms (new events) into the program whenever they become available.

Another functionality available in streaming ASP is the ability to activate and deactivate certain rules when a combination of unexpected changes in the real world affect each other. This mechanism is based on the concept of reasoning about action and change supported in the ASP semantics through default reasoning.

As an example, consider the connection between the weather condition and the traffic congestion: when traffic is heavy a user might be reluctant to take the car and go by bike instead, but the high chance of rain would need to be taken into account so that the preference of a car over a bike despite traffic congestion would be more relevant. This requires to combine preferential reasoning and default reasoning (“*in rush hour there is traffic so I would prefer to use the bike over the car, if there is little chance of rain*”).

6. Summary

In this deliverable we have documented the work carried out in the area of Adaptive Urban-Reasoning. This section summarises the current status, describes the on-going work and outlines the next steps in our work related to both TA and CF.

6.1 Outcomes

The TA functionality is achieved by identifying relevant data streams, monitoring those data stream, evaluating QoI/QoS scores after updates and trigger actions for the initiation of the adaptation request. We summarise the process below:

- **Identification of Relevant Data Streams:** the TA component receives the composition plan and event request as input, and identifies the relevant data streams to monitor for QoI/QoS updates based on information sources that are relevant for a particular application request. The identification of the relevant data streams is based on the assumption that only a single QoI/QoS update stream is generated for each data stream within the Smart City Framework.
- **Continuous Monitoring of QoI/QoS Updates:** the TA component subscribes to the identified relevant QoI/QoS updates for the data streams involved in the generation of the query response, and initiates the aggregated QoI/QoS scores evaluation algorithm after receiving any update notification.
- **Evaluation of Aggregated QoI/QoS scores of the Composition Plan:** We defined and implemented a QoI/QoS aggregation function, which can evaluate the overall QoI/QoS scores of the complete composition plan at runtime. The aggregation function considers events patterns specified within the composition plan as well as QoI/QoS metrics of the service infrastructure hosting that particular composition plan, and creates an accumulated QoI/QoS score of the composition plan. On the occurrence of any update in the QoI/QoS scores, our algorithm re-computes the accumulated QoI/QoS score and compares the threshold of the accumulated QoI/QoS constraints defined within the event request.
- **Adaptation Request Initiation:** Whenever the accumulated QoI/QoS scores after updates have fallen below the threshold, a new technical adaptation request is generated for WP3 to re-compute the composition plan. We consider only drops in QoI/QoS in order to maintain scalability of the discovery and composition process, and considering as minimum required QoI/QoS the one that is generated when the request is triggered.

In the area of adaptation via contextual filtering, we have specified and implemented adaptive functionalities that enable reaction to unexpected events and adaptation of the reasoning outcome accordingly. The approach to make this possible relies on three main steps summarised as follows:

- **Events-context correlation**: we defined an algorithm for the construction of an event graph, which correlates events and contextual elements. The creation of partitions over such graphs and the specification of an ordering relation between such partitions enable the distribution of events and contextual elements in the graph.

- **Weighting function**: based on the partitions in the graph and the way events are distributed across these partitions, we provide a mechanism to assign weights to relations (graph edges) between events and contextual elements, so that a given ordering relation between partitions is reflected in these weights.
- **Event criticality assessment**: the identification of critical events with respect to a contextual elements and a reasoning task is performed on the weighted event graph via ASP reasoning and optimization criteria for ranking the most critical events requiring adaptation at a given time.

6.2 On-going work

The adaptation request initiated by the TA component triggers a re-computation of the whole composition plan. However in some case only part of the composition plan needs to be re-computed. We are working on improving the technical adaptation process by enabling generation of technical adaptation requests within minimal modifications.

The contextual filtering capability is currently strictly related to the way events are produced by WP3. Since the event aggregation will be available at a later step, we are currently working on the generation of a set of simulated events to test the CF algorithm in the contextual travel planner scenario presented in D2.1.

Another on-going task is that of integrating our adaptation components with a 3D map for visualization and showcase of the contextual filtering functionality.

6.3 Next steps

The current implementation of the aggregated QoI/QoS score evaluation only consider accumulated value of the underlying data streams within a composition plan, we plan to further extend our algorithm to enable QoI/QoS scores evaluation at more granular level. This will enable us to apply sophisticated optimisation techniques and avoid re-computing the aggregated values of the whole composition plan whenever the QoI/QoS score of single underlying data stream is updated.

The initial formal specification and implementation of the contextual filtering capabilities is centered around each identified contextual element individually, and then combined via a specific overall weighting function. In reality, contextual elements are not always independent and they might be semantically related to each other. This correlation we plan to explore as a next step, and investigate how the weighting function can be built in such a way to take these correlations into account. As a result of this activity, the construction of the event graph might need to be dynamically adapted to such correlations and weights adjusted for better contextual filtering.

In the area of scalability, we plan to conduct deeper analysis on the impact of the transformation step between ASP format and Java. Although we expect this overhead to be minimal, further investigation is in order. Furthermore, more experiments and performance evaluations for the designed algorithms will be performed when the event tagging work from WP3 is also ready and the results of the extended methods and detailed evaluation will be reported in deliverable D5.2 and D3.3.

7. References

1. Alani, H., Kim, S., Millard, D. E., Weal, M. J., Hall, W., Lewis, P. H., & Shadbolt, N. R. (2003). Automatic ontology-based knowledge extraction from web documents. Intelligent Systems, IEEE, 18(1), 14-21
2. Alessandra Mileo, Ahmed Abdelrahman, Sean Policarpio, Manfred Hauswirth: StreamRule: A Nonmonotonic Stream Reasoning System for the Semantic Web. RR 2013: 247-252
3. ASP Competition 2011 – system track final results. https://www.mat.unical.it/aspcomp2011/SystemTrackFinalResults, 2011
4. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing, pages 263–277 (2007)
5. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press, 2003.
6. Behnel, S., Fiege, L., Muhl, G.: On quality-of-service and publish-subscribe. In: Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems. pp. 20–. ICDCSW '06, IEEE Computer Society, Washington, DC, USA (2006)
7. Bhola, S., Strom, R.E., Bagchi, S., Zhao, Y., Auerbach, J.S.: Exactly-once delivery in a content-based publish-subscribe system. In: Proceedings of the 2002 Interna- tional Conference on Dependable Systems and Networks. pp. 7–16. DSN '02, IEEE Computer Society, Washington, DC, USA (2002)
8. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: A lightweight approach for qos-aware service composition. In: Proceedings of 2nd international conference on service oriented computing (ICSOC 04) (2004)
9. Cao, F., Singh, J.: Medym: Match-early with dynamic multicast for content-based publish-subscribe networks. In: Alonso, G. (ed.) Middleware 2005, LNCS, vol. 3790, pp. 292–313. Springer Berlin Heidelberg (2005)
10. Carvalho, N., Araujo, F., Rodrigues, L.: Scalable qos-based event routing in publish-subscribe systems. In: Network Computing and Applications, Fourth IEEE International Symposium on. pp. 101–108 (2005)
11. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst. 19(3), 332–383 (Aug 2001)
12. Chen, G., & Kotz, D. (2000). A survey of context-aware mobile computing research (Vol. 1, No. 2.1, pp. 2-1). Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College
13. Cugola, G., Di Nitto, E., Fuggetta, A.: The jedi event-based infrastructure and its application to the development of the opss wfms. IEEE Trans. Softw. Eng. 27(9), 827–850 (Sep 2001)
14. E. Della Valle, S. Schlobach, M. Kro¨tzsch, A. Bozzon, S. Ceri, and I. Horrocks. Order matters! harnessing a world of orderings for reasoning over massive data. Semantic Web Journal, 2012

15. Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). The KDD process for extracting useful knowledge from volumes of data. Communications of the ACM,39(11), 27-34
16. Gama, J., Rodrigues, P. P., Spinosa, E. J., & André Carlos Ponce Leon Ferreira de Carvalho. (2010). Knowledge discovery from data streams (p. 38). Boca Raton: Chapman & Hall/CRC
17. Gao, C., Cai, M., Chen, H.: Qos-aware service composition based on tree-coded genetic algorithm. In: Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 01. pp. 361{367. COMPSAC '07, IEEE Computer Society, Washington, DC, USA (2007)
18. Gao, F., Curry, E., Bhiri, S.: Complex Event Service Provision and Composition based on Event Pattern Matchmaking. In: Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems. ACM, Mumbai, India (2014)
19. Gao, F., Curry, E., Intizar, A., Bhiri, S., Mileo, A.: QoS-aware Complex Event Service Composition and Optimization using Genetic Algorithms. In Proc. of 15th International Conference on Service Oriented Computing. Springer –Verlag. To appear.
20. GEBSER, M., KAMINSKI, R., KAUFMANN, B., OSTROWSKI, M., SCHAUB, T., AND THIELE, S. 2008. Engineering an incremental ASP solver. In Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08), M. Garcia de la Banda and E. Pontelli, LNCS vol. 5366. Springer-Verlag, 190–205.
21. Hasan, S., O'Riain, S., Curry, E.: Approximate semantic matching of heterogeneous events. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. pp. 252–263. DEBS '12, ACM, New York, NY, USA (2012)
22. Henson, C., Barnaghi, P., & Sheth, A. (2013). From Data to Actionable Knowledge: Big Data Challenges in the Web of Things. IEEE Intelligent Systems, 28(6), 0006-11
23. Iyer, R., Kleinrock, L.: "QoS control for sensor networks," Communications, 2003. ICC '03. IEEE International Conference on , vol.1, no., pp.517,521 vol.1, 11-15 May 2003
24. Le-Phuoc, D., Dao-Tran, M., Parreira, J. X., & Hauswirth, M. (2011). A native and adaptive approach for unified processing of linked streams and linked data. In The Semantic Web–ISWC 2011 (pp. 370-388). Springer Berlin Heidelberg
25. Le-Phuoc, D., Nguyen-Mau, H. Q., Parreira, J. X., & Hauswirth, M. (2012). A middleware framework for scalable management of linked streams. Web Semantics: Science, Services and Agents on the World Wide Web, 16, 42-51
26. Lijun Mei; Chan, W.K.; Tse, T. H., "An Adaptive Service Selection Approach to Service Composition,", ICWS '08. IEEE International Conference on Web Services, vol., no., pp.70,77, 23-26 Sept. 2008
27. M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, and T. Schaub. Answer set programming for stream reasoning. CoRR, abs/1301.1392, 2013
28. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP 88*, pages 1070–1080. MIT Press, Massachussets, USA, 15-18 August 1988.
29. Mahambre, S.P., Bellur, U.: An adaptive approach for ensuring reliability in event based middleware. In: Proceedings of the second international conference on Dis- tributed event-based systems. pp. 157–168. DEBS '08, ACM, New York, NY, USA (2008)
30. Mahambre, S.P., S.D., M.K., Bellur, U.: A taxonomy of qos-aware, adaptive event-dissemination middleware. IEEE Internet Computing 11(4), 35–44 (2007)

31. Martin Gebser, Benjamin Kaufmann, Torsten Schaub: Conflict-driven answer set solving: From theory to practice. Artif. Intell. 187: 52-89 (2012)
32. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub: Clingo = ASP + Control: Preliminary Report. CoRR abs/1405.3694 (2014)
33. Martin Gebser, Torsten Grote, Roland Kaminski, Torsten Schaub: Reactive Answer Set Programming. LPNMR 2011: 54-66
34. Pietzuch, P., Bacon, J.: Hermes: a distributed event-based middleware architecture. In: Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on. pp. 611–618 (2002)
35. Rowstron, A., Kermarrec, A.M., Castro, M., Druschel, P.: Scribe: The design of a large-scale event notification infrastructure. In: Crowcroft, J., Hofmann, M. (eds.) Networked Group Communication, LNCS vol. 2233, pp. 30–43. Springer Berlin Heidelberg (2001)
36. T. Eiter, G. Ianni, A. Polleres, R. Schindlauer, and H. Tompits. Reasoning with rules and ontologies. In Reasoning Web, pages 93–127, 2006
37. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Dlv-hex: Dealing with semantic web under answer-set programming. In Proc. of ISWC, 2005
38. T. Strang and C. Linnhoff-Popien. A context modeling survey. In Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004, Nottingham/England, 2004
39. Wang, F., & Liu, J. (2011). Networked wireless sensor data collection: Issues, challenges, and approaches. Communications Surveys & Tutorials, IEEE, 13(4), 673-687
40. Wu, Q., Zhu, Q., Jian, X.: Qos-aware multi-granularity service composition based on generalized component services. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) Service-Oriented Computing, LNCS vol. 8274, pp. 446–455. Springer Berlin Heidelberg (2013)
41. Wu, Q., Zhu, Q., Jian, X.: Qos-aware multi-granularity service composition based on generalized component services. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) Service-Oriented Computing, LNCS vol.8274, pp. 446{455. Springer Berlin Heidelberg (2013)
42. Zhang, C., Su, S., Chen, J.: A novel genetic algorithm for qos-aware web services selection. In: Proceedings of the Second international conference on Data Engineering Issues in E-Commerce and Services. pp. 224{235. DEECS'06, Springer-Verlag, Berlin, Heidelberg (2006)
43. Zhang, L.J., Li, B.: Requirements driven dynamic services composition for web services and grid solutions. Journal of Grid Computing 2(2), 121{140 (2004)