Automatic Configuration of Smart City Applications for User-Centric Decision Support

Thu-Le Pham*, Stefano Germano†, Alessandra Mileo‡, Daniel Küemper§ and Muhammad Intizar Ali*
*INSIGHT, National University of Ireland, IDA Bussiness Park, Lower Dangan, Galway, Ireland
Email: {thule.pham, ali.intizar}@insight-centre.org
†Department of Mathematics and Computer Science, University of Calabria, Rende, Italy
Email: germano@mat.unical.it
‡INSIGHT, Dublin City University, Glasnevin, Dublin 9, Dublin, Ireland
Email: alessandra.mileo@insight-centre.org
§University of Applied Sciences Osnabrück, Osnabrueck 49078, Germany
Email: d.kuemper@hs-osnabrueck.de

***Abstract*—Smart city applications in the Big Data era require not only techniques dedicated to dynamicity handling, but also the ability to take into account contextual information, user preferences and requirements, and real-time events to provide optimal solutions and automatic configuration for the end user. In this paper, we present a specific functionality in the design and implementation of a declarative decision support component that exploits contextual information, user preferences and requirements to automatically provide optimal configurations of smart city applications. The key property of user-centricity of our approach is achieved by enabling users to declaratively specify constraints and preferences on the solutions provided by the smart city application through the Decision Support component, and automatically map these constraints and preferences to provide optimal responses targeting user needs. We showcase the effectiveness and flexibility of our solution in two real usecase scenarios: a multimodal travel planner and a mobile parking application. All the components and algorithms described in this paper have been defined and implemented as part of the Smart City Framework CityPulse [1].**

I. Introduction

Smart city applications require Internet of Things (IoT) discovery and matchmaking techniques dedicated to dynamicity handling. Information taken into account during the matchmaking process originates from diverse data sources including: data streams, city services, the user's social context, situational awareness (e.g., user location), preferences and application configurations. The exponential growth in the availability of information from numerous data sources raises several difficulties in implementing, sustaining, and optimizing operations and interactions among different city departments and services [7]. There is a strong need for smart city application tools which support easy development of smart applications.

The state-of-the-art for smart city frameworks has major focus on existing smart city platforms and the existing works are mainly in four key areas: (i) data acquisition (ii) semantic interoperability, (iii) real-time data analysis and event detection, and (iv) smart city application development support. Among the existing frameworks such as PLAY [9], iCore [4], and STAR-CITY [5], CityPulse [8] is the only framework supporting all four previously mentioned features. In additional to data acquisition and semantic interoperability, the CityPulse framework provides a complete set of domain independent real-time data analytics tools such as data federation, data aggregation, event detection, quality analysis and decision support. The application development is supported through a set of APIs provided by CityPulse. In this paper, we focus on the decision making process, which is designed and implemented within CityPulse framework.

The Decision Support component in CityPusle supports the complex reasoning capabilities that are required in various smart city applications such as non-monotonic, non-deterministic, and recursive reasoning. This component represents higher-level intelligence, strongly connects to user application layer, and acts as a flexible interplay between user-centric factors and dynamic aspects of the changing environment within the city. Factors such as user interests and reputation requirements are also considered in the decision support process. The exploitation of such factors along with richer user profiles has a great potential for providing more personalised decision support, and greatly improve user experience when interacting with smart city applications. Although elicitation and usage of user profiles is optional, motivated by their potential we included explicit aspects of user profiles in the decision support process which are automatically encoded and used to configure the way the decision support process works. These aspects include not only user location but also user preferences and constraints on the solutions provided, as well as dynamic correlations between contextual activities and their dependencies with city events for a particular user in specific application scenarios.

The ability to continuously characterize the correlation between city events and user activities is used to dynamically filter events that are relevant for a particular user at a specific time so that Decision Support can be instructed to find new solutions whenever needed. This functionality has been specified and implemented in a component called Contextual Filtering [8]. In the proposed characterisation, events, user activities and

[1] www.ict-citypulse.eu

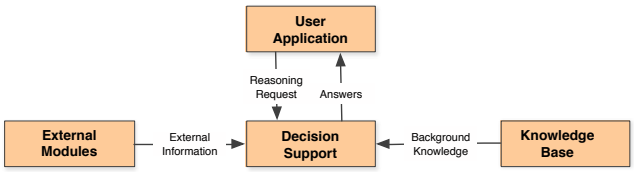

Fig. 1. Decision Support I/O

their dependencies are modelled using Linked Data and open vocabularies in order to provide a lightweight, interoperable and well-established foundation for decision support.

In this paper we focus on user-centricity and describe how user requirements in terms of preferences and constraints can be explicitly specified and mapped into a representation that is independent from the specific application.

In the remainder of this paper we will present the design principles for our Decision Support component in Section II, formalise our declarative specification of user requirements and their automatic representation as logical rules in Section III, and the implementation of our scenarios in Section IV. We then conclude with some final remarks in Section V.

II. User-Centric Decision Support

The Decision Support component is responsible for higher-level intelligence, which can utilize user contextual information, background knowledge, and real-time events to deduce intelligent conclusion in real-time. This component is also capable of acquiring the analyzing additional information sources related to user contextual patterns, users' application usage behaviour, and self-defined preferences (while using a smart city application) to provide optimal configuration for smart city applications and enable these applications to generate reactive application logic within deployed smart city applications. Figure 1 represents a general information flow and interactions of Decision Support with other external components.

As an initial step to start the information processing, Decision Support receives following input: (i) *a reasoning request* from the application interface which includes user related functional and non-functional parameters, constraints, and preferences, (ii) *background knowledge*, which is domain dependent information available for reasoning and application logic strictly tied to the given scenario, and (iii) *external information sources*, which is any relevant information collected through external components required for that specific scenario. After processing all related input information as mentioned above, Decision Support generates a set of scenario-driven solutions, which are guaranteed to be optimal and satisfying all requirement and preferences specified by the individual user.

In real world scenarios, the reasoning module has to deal with issues related to incomplete and contradictory information, diverse and un-realiable input data, and most importantly user defined constrained and preferences are not explicitly input by the end user. In order to better support the provision of optimal decisions, the reasoning module must have ability to expressively deduce information from the information collected through internal and external modules additional to the user defined input. We achieved this expressivity within Decision Support by opting to use a declarative non-monotonic logic reasoning approach based on Answer Set Programming (ASP) [6]. All input information of Decision Support is mapped into ASP-format rules. Decision Support combines mapped rules with already existent domain-dependent rules, to design an application logic for the provision of optimal solutions to the users. Figure 2 depicts a system sequence diagram of Decision Support to showcase all interactions and processing steps involved in this component. In what follows, we briefly elaborate each and every information processing step of the Decision Support component.

- *Request Handler* receives a *ReasoningRequest* as an input from the application containing user preferences and requirements. Whenever a new reasoning request is arrived, a new instance of Decision Support is initiated. This ReasoningRequest is interpreted as the *InterpretedRequest* and used to initiate the *DS Manger*.
- *Request Re-writer* automatically generates the logic rules required for the specific reasoning request that is received from DS Manager, the detailed process of automated mapping and rules generated is presented in Section III. After receiving the rules from Request Rewriter, the DS Manager asks *CoreEngine* to perform the reasoning by sending InterpretedRequest and Rules as parameters.
- *CoreEngine* is a component that executes the ASP solver (in our current implementation, we use Clingo4 [3] as the ASP solver) using the EmbASP framework[2] which is able to invoke the ASP solver and to collect Answer Sets as plain Java objects. The ASP solver starts by collecting:
  - *ExternalInformation* refers to additional information collected through external modules, which can vary depending on the scenario. For example, in the Travel Planner scenario (see Section IV), the possible routes from starting point to ending point or the latest city events can be considered as *ExternalInformation*. Decision Support directly interacts with external modules within the ASP program, using a technique called the "external atoms". By using this technique, the ASP reasoner is able to invoke the external modules interactively, only on need basis and can also re-use derived answers for other reasoning tasks. This feature offers a very powerful ability on-demand composition of reasoning tasks to provide solutions.
  - *BackgroundKnowledge* is static information containing important facts and rules related to a particular domain and stored internally for reasoning tasks. For example, in the Parking Space scenario (see Section IV), the locations of the parking spaces are part of background knowledge.

[2]https://www.mat.unical.it/calimeri/projects/embasp/

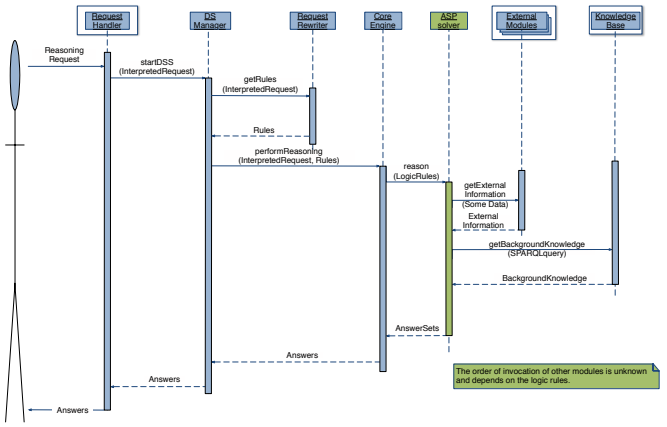

Fig. 2. Decision Support sequence diagram

- The ASP Solver combines *ExternalInformation*, *BackgroundKnowedgle*, and *Rules* to compute the optimal answers (in the form of AnswerSets) satisfying users' defined constraints and preferences.
- *EmbASP Framework* processes AnswerSets and the optimal selected *Answers* (as objects) are sent to the CoreEngine and then back to the user application for visualization.

The actual deduction process for generating solutions to the decision support task required by the application is performed by combining background knowledge, external information, user preferences & constraints and scenario-dependent rules. The fully declarative nature of the ASP framework used in the implementation of the Decision Support component capabilities makes it possible to combine these rules and knowledge facts in a straightforward way, and enables full exploitation of the expressive power of ASP inference for constraint checking and preference-based deduction. This very same declarative feature is likely to simplify the extension of Decision Support provided within the CityPulse framework and to reuse for other decision support tasks.

III. User-Centric Decision Support Request: Specification and Mapping

Preference-driven and constraint-based reasoning provides a powerful mechanism where user-centricity is a key feature, and enables to find an optimal match between the needs, preferences of citizens, available data streams and city services. This section describes the specification of a reasoning request. We focus on user preferences & constraints and their automatic mapping into declarative deduction rules. These rules are then used in the decision support process for solution optimization.

In what follows we will detail each element of the Reasoning Request and define the automatic mapping or translation into deduction rules used by the Decision Support component. Such translation is defined in a general way so that independently of the Functional Details defined as strings and values, an automatic declarative rule-based specification can be obtained, which is seamlessly combined with the rules in the Decision Support module used by a specific application.

As illustrated in Figure 3, the Reasoning Request consists of:

- **Type** (T) determines the reasoning task required by the application. This is used directly by Decision Support to perform the correct task using the reasoning engine, and needs to be identified among a set of available options at design time by the application developer. Such options have been defined for the implemented scenarios. Customization and extension of available types will be possible via APIs.
- **User Reference** is an identifier of the user that made the request. Such reference is meant to be a unique identifier 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.
- **Functional Details** represent the actual criteria for the reasoning task required by the user (i.e. constraints and preferences for solution optimization). Functional Details includes Functional Parameters, Functional Constraints, and Functional Preferences.

We shall focus now on the specification of each of the aspects included in Functional Details, and illustrate how they are automatically mapped and translated into logical deduction rules.

A. Functional Parameters

A Functional Parameter defines a mandatory information for the Reasoning Request (for instance the "ending point" in a travel planner scenario). A set of Functional Parameters (Π) is composed by a finite set (of cardinality n_Π) of individual Functional Parameter ($\pi^i; \pi^i \in \Pi \ \forall i \in [1; n_\Pi]$). Each Functional Parameter (π^i) is composed of:

- Functional Parameter Name (N_{π^i})
- Functional Parameter Value (V_{π^i})

i.e. $\pi^i = \langle N_{\pi^i}, V_{\pi^i} \rangle$.

The Functional Parameter Name (N_{π^i}) is a string taken from a fixed set of strings ($\Theta_{T,N_{\pi^i}}$) and the Functional Parameter Value (V_{π^i}) is specific for each scenario.

The set of Functional Parameters (Π) is translated as the concatenation of the translations of each Functional Parameter (π^i) that composes it. Each Functional Parameter ($\pi^i = \langle N_{\pi^i}, V_{\pi^i} \rangle$) is translated as:

$$parameter(N_{\pi^i}, V_{\pi^i}).$$

The Functional Parameter Value can be a single value or a set of possible values. When the Functional Parameter Value is a set (e.g. expressed as enumeration of possible values), it is translated into several of the above facts, one for each item in the set.

B. Functional Constraints

A Functional Constraint defines a numerical restriction about a specific aspect of the Reasoning Request. This restriction is "strict" and needs to be fulfilled by each of the answers (otherwise referred to as solutions) offered to the user. A set

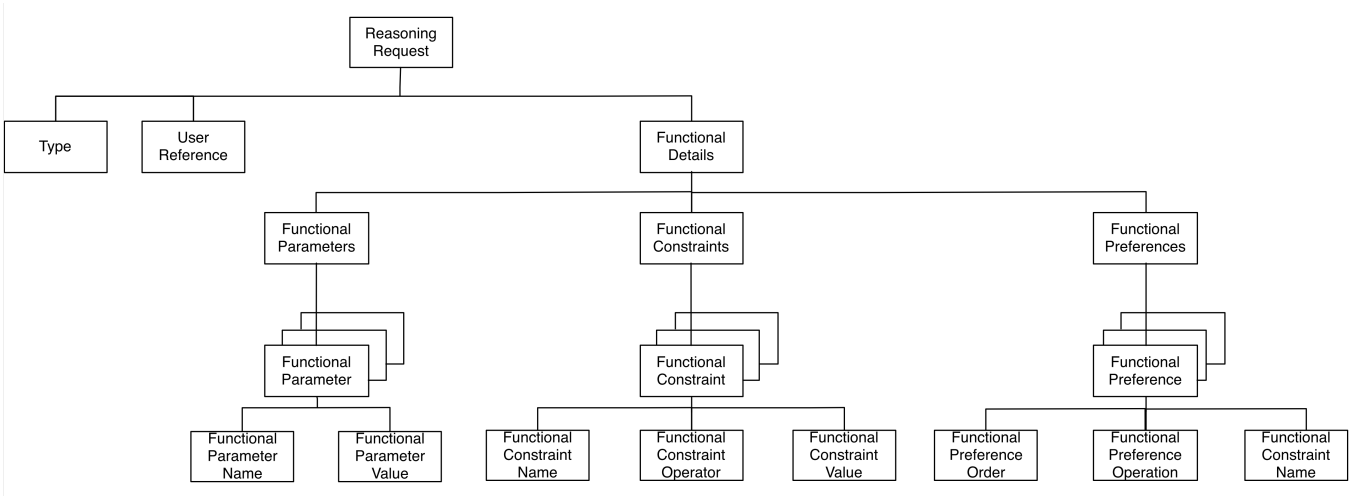

Fig. 3. Representation of Reasoning Request

of Functional Constraints (Γ) is composed by a finite set (of cardinality n_Γ) of individual Functional Constraint ($\gamma^i; \gamma^i \in \Gamma \; \forall i \in [1; n_\Gamma]$).

Each Functional Constraint (γ^i) is composed of:

- Functional Constraint Name (N_{γ^i})
- Functional Constraint Operator (O_{γ^i})
- Functional Constraint Value (V_γ)

i.e. $\gamma^i = \langle N_{\gamma^i}, O_{\gamma^i}, V_\gamma \rangle$.

The Functional Constraint Name is a string taken from a fixed set of strings ($\Theta_{T,N_{\gamma^i}}$) and the Functional Constraint Operator is an arithmetic operator taken from a fixed set ($\Theta_{O_{\gamma^i}} = \{=, \neq, >, <, \geq, \leq\}$). For each Functional Constraint Operator (O_{γ^i}) we denote with $\overline{O_{\gamma^i}}$ its complementary operator. The Functional Constraint Value V_γ is an integer number.

The Functional Constraints (Γ) is translated as the concatenation of the translations of each Functional Constraint (γ^i) that composes it. Each Functional Constraint is translated as:

- The "real" constraint:

$$\leftarrow violatedC(N_{\gamma^i}).$$

- A rule to derive if it is violated:

$$violatedC(N_{\gamma^i}) \leftarrow valueOf(N_{\gamma^i}, AV), AV\overline{O_{\gamma^i}}V_{\gamma^i}.$$

where AV is an ASP variable.

C. Functional Preferences

A Functional Preference defines a "soft" constraint or priority among the verification of specific aspect of the Reasoning Request. This restriction is "weak" and should be optimized by the Decision Support component in order to provide the optimal or most preferred answers to the user.

A set of Functional Preferences (Ω) is composed by a finite set (of cardinality n_Ω) of individual Functional Preference ($\omega^i; \omega^i \in \Omega \; \forall i \in [1; n_\Omega]$). Each Functional Preference (ω^i) is composed of:

- Functional Preference Order (O_{ω^i}).
- Functional Preference Operation (Opt_{ω^i})
- Functional Constraint Name (N_{γ^i}) is defined in III-B.

i.e. $\omega^i = \langle O_{\omega^i}, Opt_{\omega^i}, N_{\gamma^i} \rangle$.

The Functional Preference Order (O_{ω^i}) is an integer $\in [1; n_\Omega]$ and the Functional Preference Operation (Opt_{ω^i}) is an optimization operator taken from a fixed set ($\Theta_{Opt_{\omega^i}} = \{minimize, maximize\}$). The Functional Constraint Name (N_{γ^i}) is defined in III-B.

The Functional Preferences (Ω) is translated as the concatenation of the translations of each Functional Preference (ω^i) that composes it. Each Functional Preference is translated as:

$$\#Opt_{\omega^i}\{AV@O_{\omega^i} : valueOf(N_{\gamma^i}, AV)\}.$$

To allow more flexibility in the logic program each Functional Preference (ω^i) could be also translated as (in addition to the previous translation):

$$preference(O_{\omega^i}, Opt_{\omega^i}, N_{\gamma^i}).$$

IV. Usecase Scenarios

In order to demonstrate how Decision Support can be used to develop applications for smart cities and citizens, we have implemented two context-aware usecases using the live data from the city of Aarhus, Denmark: a Travel Planner app and a Parking Planner app . In this section, we present the Reasoning Request, logic rules automatically generated from the Reasoning Request, scenario-dependent rules, and External Modules used in the decision support process.

A. Travel Planner Scenario

Tony needs to travel from home to work. Different means of transportation are generally available to him and include walking, biking, car, and public transport. Transportation can be optimized to Tony's preferred travel time, convenience, total cost, environmental impacts, and personal health. Factors that impact this optimization include the conditions of the different transportation modes, including but not limited to road, weather, maintenance works, traffic intensity, people density, pollution, air quality, irregularities in traffic schedules, road tolls, seating availability, accidents, availability of city

bikes. Tony will be presented with his ideal route and will be able to select each leg of the journey based on concurrent and projected aggregated conditions. Recalculation of his chosen route(s) can happen if conditions or preferences change, and the provided solution will adapt to any detour of own choice.

In order to provide optimal travel-planning solutions to Tony, Decision Support allows him to provide his multi-dimensional requirements and preferences such as air quality, traffic conditions, ect. The Reasoning Request for this scenario has the following main fields:

- Type: indicating what decision support module of the Smart City Framework is to be used for this application ("TRAVEL-PLANNER" in this case).
- User Reference: indicating the unique userId.
- Functional Details: specifying possible values of user's requirements and preferences. Tables I, II, and III show concrete possible values of Functional Parameters, Functional Constraints, and Functional Preferences respectively.

TABLE I
EXAMPLE OF FUNCTIONAL PARAMETERS FOR THE TRAVEL PLANNER SCENARIO

Functional Parameters	Name	Value Type	Value
Starting Point	STARTING_POINT	Coordinate	56.17888121694039 10.153993614949286
Ending Point	ENDING_POINT	Coordinate	56.15183187883248 10.154508599080145
Starting Time/Date	STARTING_DATETIME	Date	2017-01-10T18:25:43.511Z
Transportation Type	TRANSPORTATION_TYPE	Enum	{CAR, WALK, BICYCLE}

TABLE II
EXAMPLE OF FUNCTIONAL CONSTRAINTS FOR THE TRAVEL PLANNER SCENARIO

Functional Constraints	Name	Operator	Value Type	Value
Travel time less than X	TRAVEL_TIME	$\leqslant$	Duration	15
Distance less than X	DISTANCE	$\leqslant$	Number	1000
Pollution amount less than X	POLLUTION	$\leqslant$	Number	13,5

TABLE III
EXAMPLE OF FUNCTIONAL PREFERENCES FOR THE TRAVEL PLANNER SCENARIO

Functional Preferences	Order	Operation	Name
Travel time	1	MINIMIZE	TRAVEL_TIME
Distance	2	MINIMIZE	DISTANCE
Pollution	3	MINIMIZE	POLLUTION

The concrete reasoning request is automatically mapped into ASP rules (see example rules 1-8 in Listing 1) in which: Functional Parameters are translated as simple logic facts (rules 1-2); Functional Constraints (rules 3-4) are translated as strong constraints, which reduce the solution space by eliminating answers that are violating any of those constraints. Functional Preferences are translated as optimize statements (rules 5-8), which rank the solutions to provide only those that are qualitatively better with respect to the optimization statements used. Those rules are combined with the specific scenario-driven rules for the Travel Planner Decision Support module (rules 9-13) for reasoning[3]. The Decision Support component collects all possible routes from the Geo-spatial Database Infrastructure (GDI) component [8] 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 [2], [1], [8] or retrieved from the Knowledge Base (rules 10-12).

The External Modules used for this scenario are:

- GDI, which enables calculation of different distance measures and allows enhanced information interpolation to increase reliability. Furthermore, an enhanced routing system enables multidimensional weighting on path, e.g., depending on distance, duration, pollution, events or combined metrics. Thereby, it is possible to avoid certain areas or block partial routes for specific applications.
- Data Federation, which is responsible for processing the application request for IoT streams and automatically discover the most relevant data streams after catering for individual requirements and preferences for a particular user request. It is also responsible for automatically integrating heterogeneous data streams and perform complex event processing over the integrated stream.

Both the GDI and the Data Federation components are part of the CityPulse framework. Their implementation are available at https://github.com/CityPulse.

```
1. parameter("ENDING_POINT","10.1591864 56.1481156").
2. parameter("STARTING_POINT","10.116919 56.226144").
3. :- violatedConstraint("POLLUTION").
4. violatedConstraint("POLLUTION") :- valueOf("POLLUTION",
     AV), 135 < AV.
5. #minimize{AV@1 : valueOf("TIME", AV)}.
6. preference(1,"MINIMIZE","TIME").
7. #minimize{AV@2 : valueOf("DISTANCE", AV)}
8. preference(2,"MINIMIZE","DISTANCE").
9. input_get_routes(SP, EP, V, 5) :- parameter("
     STARTING_POINT", SP), parameter("ENDING_POINT", EP),
     route_costMode(V).
10. route(@get_routes(SP, EP, V, N)) :- input_get_routes(SP
     , EP, V, N).
11. route_data(@get_routes_data(SP, EP, V, N)) :-
     input_get_routes(SP, EP, V, N).
12. max_pollution(@get_max_pollution(RouteID)) :- selected(
     RouteID).
13. 1 <= {selected(RouteID) : route((RouteID, _, _))} <= 1.
```

Listing 1. A snapshot of logic rules for Travel Planner scenario

B. Parking Planner Scenario

Frank is having a hard time finding a public parking space. The city is increasingly reducing the amount of parking spaces per unit (e.g. apartments), and the difficulty of finding a parking space means Frank has to drive around for a long time looking for parking spots. This is very time consuming for him and results in negative environmental impact (pollution, noise). By using multiple input sources of information the application can provide Frank with a certain degree of probability of finding a parking spot in different locations, thus reducing the driving time (and related CO2 emissions). By knowing the number of cars on the road at any time, the application can help Frank avoiding congested hot spots by being rerouted towards different paths to even out the distribution.

Decision Support aims to provide optimal available parking slots nearby Frank's point of interest while taking into account

[3]A full set of rules is available at https://github.com/CityPulse/Decision-Support-and-Contextual-Filtering/tree/master/res/dss

TABLE IV
EXAMPLE OF FUNCTIONAL PARAMETERS FOR THE PARKING PLANNER SCENARIO

Functional Parameters	Name	Value Type	Value
Starting Point	STARTING_POINT	Coordinate	56.17888121694039 10.153993614949286
Point Of Interest	POINT_OF_INTEREST	Coordinate	56.15183187883248 10.154508599080145
Starting Time/Date	STARTING_DATETIME	Date	2017-01-10T18:25:43.511Z
Distance Range	DISTANCE_RANGE	Number	1000
Time Of Stay	TIME_OF_STAY	Duration	100

TABLE V
EXAMPLE OF FUNCTIONAL CONSTRAINTS FOR THE PARKING PLANNER SCENARIO

Functional Constraints	Name	Operator	Value Type	Value
Walking distance less than X	DISTANCE	$\leqslant$	Number	1000
Cost less than X	COST	$\leqslant$	Number	50

his constraints and preferences. The Reasoning Request for this scenario has the following main fields:

- Type: indicating what decision support module is to be used for this application ("PARKING-SPACE" in this case).
- User Reference: indicating the unique userId.
- Functional Details: specifying possible values of user's requirements and preferences. Tables IV, V, and VI show concrete possible values of Functional Parameters, Functional Constraints, and Functional Preferences respectively.

```
1. parameter("DISTANCE_RANGE",1000).
2. parameter("POINT_OF_INTEREST","10.116919 56.226144").
3. :- violatedConstraint("COST").
4. violatedConstraint("COST") :- valueOf("COST", AV), 100 <
     AV.
5. preference(1,"MINIMIZE","DISTANCE").
6. #minimize{AV@1 : valueOf("DISTANCE", AV)}.
7. preference(2,"MINIMIZE","COST").
8. #minimize{AV@2 : valueOf("COST", AV)}.
9. parking_space(@get_parking_spaces(POI, DR)) :- parameter
     ("POINT_OF_INTEREST", POI), parameter("DISTANCE_RANGE",
     DR).
10. availability(@get_availability(ParkingID)) :- selected(
     ParkingID).
11. total_cost(@get_total_cost(ParkingID, ToS)) :- selected
     (ParkingID), parameter("TIME_OF_STAY", ToS).
12. 1 <= {selected(ParkingID) : parking_space((ParkingID,
     Position,Distance))} <= 1.
13. distance(Distance) :- selected(ParkingID),
     parking_space((ParkingID,Position,Distance)).
```

Listing 2. A snapshot of logic rules for Parking Planner scenario

Similar to the Travel Planner scenario, the concrete reasoning request is automatically mapped into ASP rules (see example rules 1-8 in Listing 2), and combined with the specific scenario-driven rules for the Parking Decision Support module (rules 9-13). The Decision Support component collects all possible parking slots with their cost from the Knowledge Base (these parking slots are in 'DISTANCE_RANGE' which is checked by resorting to the GDI component) as well as the last snapshot of availability of parking slots which can be produced dynamically by the Data Federation component (rules 9-11). Similarly to the Travel Planner scenario, the External Modules used for this scenario are GDI and Data Federation.

TABLE VI
EXAMPLE OF FUNCTIONAL PREFERENCES FOR THE PARKING PLANNER SCENARIO

Functional Preferences	Order	Operation	Name
Walking Distance	1	MINIMIZE	DISTANCE
Cost	2	MINIMIZE	COST

V. CONCLUSION

In this paper, we describe how we designed and implemented a user-centric declarative Decision Support component by leveraging the expressivity and fully declarative nature of ASP. To achieve this, we define a representation method that allows a user to specify constraints and preferences, and we propose an automatic mapping to convert user requests into logical rules. In order to demonstrate the efficiency in term of reusability and declarativity of this approach, we showcase the implementation of the Decision Support component for two smart city applications: the Travel Planner and the Parking Planner applications. Our rule-based Decision Support component can be used in various application scenarios by: (i) identifying values of the parameters to be constrained or optimized in the Reasoning Request, (ii) describing the domain-specific rules for the decision task (or using existing reasoning modules available in the CityPulse Framework), (iii) plugging in the proper External Modules to compute subtasks when needed for scalability. As a result of our proposal, we can achieve user-centricity in the Decision Support process in order to provide optimal solutions that better target user needs.

ACKNOWLEDGMENT

This research has been partially supported by SFI under grant No. SFI/12/RC/2289 and the EU FP7 CityPulse Project under grant No.603095. http://www.ict-citypulse.eu.

REFERENCES

[1] F. Gao, M. I. Ali, E. Curry, and A. Mileo. Qos-aware adaptation for complex event service. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1597–1604. ACM, 2016.

[2] F. Gao, E. Curry, M. I. Ali, S. Bhiri, and A. Mileo. Qos-aware complex event service composition and optimization using genetic algorithms. In *International Conference on Service-Oriented Computing*, pages 386–393. Springer, 2014.

[3] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Clingo= asp+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*, 2014.

[4] R. Giaffreda. icore: a cognitive management framework for the internet of things. In *The Future Internet Assembly*, pages 350–352. Springer, 2013.

[5] F. Lécué, S. Tallevi-Diotallevi, J. Hayes, R. Tucker, V. Bicer, M. Sbodio, and P. Tommasi. Smart traffic analytics in the semantic web with star-city: Scenarios, system and lessons learned in dublin city. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27:26–33, 2014.

[6] V. Lifschitz. What is answer set programming?. In *AAAI*, volume 8, pages 1594–1597, 2008.

[7] M. Naphade, G. Banavar, C. Harrison, J. Paraszczak, and R. Morris. Smarter cities and their innovation challenges. *Computer*, 44(6):32–39, 2011.

[8] D. Puiu, P. Barnaghi, R. Tönjes, D. Kümper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. Farajidavar, et al. Citypulse: Large scale data analytics framework for smart cities. *IEEE Access*, 4:1086–1108, 2016.

[9] R. Stühmer, Y. Verginadis, I. Alshabani, T. Morsellino, and A. Aversa. Play: Semantics-based event marketplace. In *Working Conference on Virtual Enterprises*, pages 699–707. Springer, 2013.