A Conceptual Framework for Development of Context-aware Location-based Services on Smart-M3 platform

Ilya Paramonov*†, Andrey Vasilyev*†, Eldar Mamedov†
{ilya.paramonov,andrey.vasilyev}@fruct.org, eldar.mamedov@e-werest.org
*Petrozavodsk State University, Petrozavodsk, Russia
†P.G. Demidov Yaroslavl State University, Yaroslavl, Russia

***Abstract*—The paper presents a conceptual framework for development of context-aware location-based services. This framework provides relevant objects from the database to the user taking his/her preferences and context into account. It is based on the framework for context-aware preference queries, which provides a model of context- and preference-aware system based on the database, and the open source Smart-M3 platform, which allows to develop intelligent services using the smart space paradigm. The main components of the proposed framework include context-aware preference term generators that translate context information into context-aware preference terms, and a preference query executor that combines all preference terms and conducts their execution using PreferenceSQL JDBC driver. Evaluation of the proposed approach is made using the case study of context-aware restaurant data retrieval.**

I. Introduction

Location-based services (LBS) are services oriented at mobile users that take current position of the user into account when performing their tasks. They occupy a niche at the intersection of geographic information systems (GIS), Internet, and mobile devices making use of the locationing facilities of the latter [1].

Context can be considered as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves" [2].

The LBS themselves are based on taking into account a particular part of the context, the location. But considering other aspects of context into account can provide more relevant information and appropriate services for customers [3]. For example, routing for cars service benefits from access to the traffic conditions and routing for people benefits from knowing the public transport location [4].

Smart-M3 is an open source platform for development of intelligent services based on the smart space paradigm [5]. In Smart-M3 different services can communicate via shared memory and use the subscription mechanism to get notifications on data modifications. This approach allows services to seamlessly share parts of the context or use context data provided by other parties.

In this paper we propose a conceptual framework intended to develop context-aware location-based services for Smart-M3 platform. On a low level it is based on a framework for context-aware preference queries [6], whereas on a high level it uses openness of the Smart-M3 platform to allow online addition or removal of context-aware generators providing various intelligence levels for user experience and subscription mechanism to integrate all components to the system.

The rest of the paper is structured as follows. Section II overviews related works about relationship between context-awareness, LBS, and smart spaces. Section III informally presents the preference model used in this paper and describes context-aware preference generation principles. Thereafter, Section IV provides an architecture of the entire framework, its data structures and describes its main components. Section V contains a case study that allows to estimate usefulness of the proposed framework. Conclusion summarizes the main results of the paper.

II. Related work

There are no papers that propose a holistic framework for development of context-aware location-based services for Smart-M3 platform. However, the papers simultaneously covering two of these three main topics (context-awareness, LBS, and services for Smart-M3 platform) can be of particular interest for the purpose of our study.

Researchers from the University of Minnesota introduced the system architecture of a Context and Preference-Aware Location-based Database Server (CareDB) that provides personalized services to its customers based on the context [7]. CareDB allows context and preference-aware query processing using the internal database and external services including LBS platform. The researchers identify various types of context and a plethora of multi-objective preference methods that are capable to evaluate user preference constraints.

In [8] the authors present the SHERLOCK system that processes user requests continuously to provide up-to-date answers in heterogeneous and dynamic contexts. It offers a user an access to a set of interesting LBS. When the user selects the desired category the system automatically determines aspects of his/her information needs via query to the local knowledge base. This relives the user from the need to manually interact with the plethora of location-based services.

In [9] the authors propose a model of the context-based access control to data shared in a smart space. The model is built on the combination of the role-based and attribute-based

access control models. In this model the participant's context is used to define the trust levels and assign a particular role. The context includes identification attributes, location, current date, device type, etc.

In [10] the authors present an approach for developing context-aware intelligent applications for smart space-based infrastructure. They propose a context reasoning to infer high level context. The inference rules are used to form new context related to an individual and can define how to react to different situations. The authors defined a such rule using a 3-clauses pattern: *with*-clause, *when*-clause, and *then*-clause.

In [11] the authors introduce the methodology and design of the SmartRoom system that support collaboration activities localized in a room. It simplifies the multi-party activity organization and human participation by automating routine functions of information acquirement, sharing, and transformation. The SmartRoom service set is developed on top of the Smart-M3 platform. This system provides facilities to store and use a context of users.

In [12] the authors propose an approach to create a proactive location-based service upon combination of platforms for smart space creation (Smart-M3) and geo-tagging (Geo2Tag). The use of geographical and temporal data in the smart space gives developers an ability to identify object's location in the real world and query objects based on their position.

The mobile ridesharing logistics solution using the Smart-M3 platform is presented in [13]. The main idea of the system is to provide models and methods that would enable configuration of resources for decision support in ad-hoc sustainable logistics. In the proposed approach the dynamic ridesharing for passengers as well as for cargo is considered.

In [14] the authors propose an location-based application "Tourist assistant–TAIS". It is an intelligent mobile tourist guide that recommends the tourist attractions near by based on his/her preferences and context. This application has been developed on top of the Smart-M3 platform. It consists of a set of services that interact with each other for providing the tourist recommendations about attraction that is better to see around.

III. Context awareness in location-based services

A. Preference model

In our research we use the preference model presented in [15]. This preference model is used in the Preference SQL[1], which is a well-established framework to create personalized information systems [16]. The model defines preferences in a way that is suitable for translation into relational algebra statements. It allows to retrieve objects in the most suitable for the end-user order taking context information into account.

Following [15], we consider a *preference* as a *strict partial order* on the domain values of attributes of the database relation. The result of a preference is computed by the *preference selection*, which are all tuples from a database relation that are maximal according to the preference order.

[1] http://preferencesql.com

To specify a preference, a number of base constructors and two complex constructors can be used.

Base preferences are preferences defined on a single attribute. There are base preference constructors for *continuous*, *discrete (categorical)* and *spatial* domains. Here are examples of base preference constructors:

- *POS preference*. The discrete Positive-preference POS states that the user has a set of preferred values.
- *BETWEEN preference*. The continuous preference constructor BETWEEN expresses the wish for a value between a lower and an upper bound. If this is infeasible, values having the smallest distance to a lower and an upper bound are preferred.
- *LESS_THAN preference*. The continuous preference constructor LESS_THAN expresses the wish for a value lower than specified bound.
- *LOWEST preference*. The continuous preference constructor LOWEST expresses the wish for a lowest value of an attribute.

In [17] there are also several constructors for spatial query support described:

- *WITHIN preference*. The spatial preference constructor WITHIN expresses the wish for a geographical objects that are within or close to a region. A first object is better than second if distance from it to the region is less than distance from the second object to the same region.
- *NEARBY preference*. The spatial preference constructor NEARBY is the same as WITHIN but differs in that NEARBY accepts a point instead of a region.
- *ONROUTE preference*. The spatial preference constructor ONROUTE is the same as WITHIN but differs in that ONROUTE accepts a set of points that represent a route instead of a region.
- *SCORE preference*. The SCORE preference is the parent of all base preference constructors, it allows to specify a preference using a numerical scoring function. Basically, it is not a spatial preference, but can be used for spatial queries when using a distance-related scoring function.

Complex preference describes the ranking between a set of preferences. They include:

- *Pareto preference*. This preference represents equal importance between several preferences. It is denoted by the $\otimes$ sign.
- *Prioritization preference*. This preference represents "greater than" importance. It is denoted by the & sign. In a Prioritization preference the importance of the preferences decreases from left to right.

With the use of complex preferences it becomes possible to state compound requirements on top of the data query.

B. Context model

In this paper, we consider context as a set of properties related to the user and representing his/her state. A context property is a pair of a key and an associated value. The context representation may be complex and include several objects that relate to each other in some way, but in current research we intentionally use simple structure to describe the context.

There are two types of user context, namely, the *static user context* and the *dynamic user context*.

The *static user context* contains description of the user that rarely changes. These data are usually stays the same throughout series of requests. For example, the static user context may include user's profession, age, sex, physique, income, etc. Partially this context can be provided by the user and can be changed whenever she wants.

The *dynamic user context* contains user description that frequently changes. These data could be different for each request and may change during the request processing, therefore it is provided as a part of the request. An example of the dynamic user context is the current user location.

C. Context-aware generators

The context properties provide facts about the user while preference terms define order, limits, and restrictions onto the domain of the queried objects. To fill this gap authors in [6] present context-aware generators that can translate data about users into domain-specific preferences. In order to do so they must be aware of both domains and their relations.

In the simplest case the generator can take a value from a context property and place it into the corresponding preferences template. For example, if a person has children and searches for a place to rest during holidays, then generator can provide preferences to search for facilities that provide special services for families. More sophisticated generators can transform context values according to predefined rules in some complex way or query external services to retrieve sorting parameters. For example, if a person uses online service to rate movies one watched and searches for a themed party to visit, the generator can retrieve top-5 interests from the service and form a POS preference.

IV. Framework architecture

A. Smart-M3 interoperability platform

Smart-M3 is an open-source platform that provides facilities for creation of the multi-agent distributed applications with the shared view of dynamic knowledge and services for ubiquitous computing environments [5]. It operates on principles of transparent data exchange between services that gives developers a way to build open systems that can be easily extended.

The platform consists of two main components: a semantic information broker (SIB) and a set of services called knowledge processors (KPs). The SIB is the core component that stores shared semantic data. KPs are active parts of a system, they can provide modify and query data using the insert, remove, update, query, and subscribe operations provided by the SIB. The shared data describe the state of the smart space and interests of the involved parties.

The data in the SIB is stored in the form of a Resource Description Framework[2] (RDF) graph, usually according to some defined ontology. This framework is based on triples, each of which consists of a subject, a predicate, and an object. The triples usually represent different objects and their relations between one another. The Web Ontology Language (OWL, [18]) is commonly used to describe object classes and meaning of the object properties.

The communication mechanism between KPs and SIB is called the smart space access protocol (SSAP). It has eight operations: join, leave, insert, remove, update, query, subscribe, and unsubscribe. Subscription operation allows a KP to persistently query a part of the common data graph. When the tracked part of the graph changes its state, SIB sends a corresponding notification.

Our framework is targeted at the Smart-M3 platform. All data about the users are stored in the shared storage, all user requests are handled using Smart-M3 mechanisms, and the main parts of the framework are implemented as KPs communicating via the data modification in SIB.

B. Framework overview

The purpose of the proposed framework is to provide relevant objects from the database to client applications in consideration with user preferences and context. The retrieved objects not only conform to the request but have a particular order by descending relevance. The components of the framework and the data flow between them are shown in Fig. 1.

In order to facilitate the desired behavior the framework uses Preference SQL terms during query of required objects from the database. These terms allow to present different complex relationship and constraints, their mathematical counterparts were presented in Section III-A. They are created and combined into resulting term from user preferences and context according to predefined rules.

The framework considers three different kinds of input that affect the order of retrieved objects: the request from the client service, the corresponding user's context, and the profile. To generate the combined preference term the framework uses a composition according to [6]. The parameters of the request have the greatest influence on the result, the current context has the next priority, and the default preferences have the least influence on the query. The combination of inputs using prioritization composition is the following:

<preference_based_query> := <client_request> &
<user_context> & <user_profile>

The client requests objects in the form of preference terms. The user profile is also a set of preference terms that describe common requirements from the user for each type of searchable objects. The client application may update last set of terms at any time.

The user context is initially a set of properties. In order to create a preference term from the context there are special entities called context-aware preference term generators

[2]http://www.w3.org/RDF/

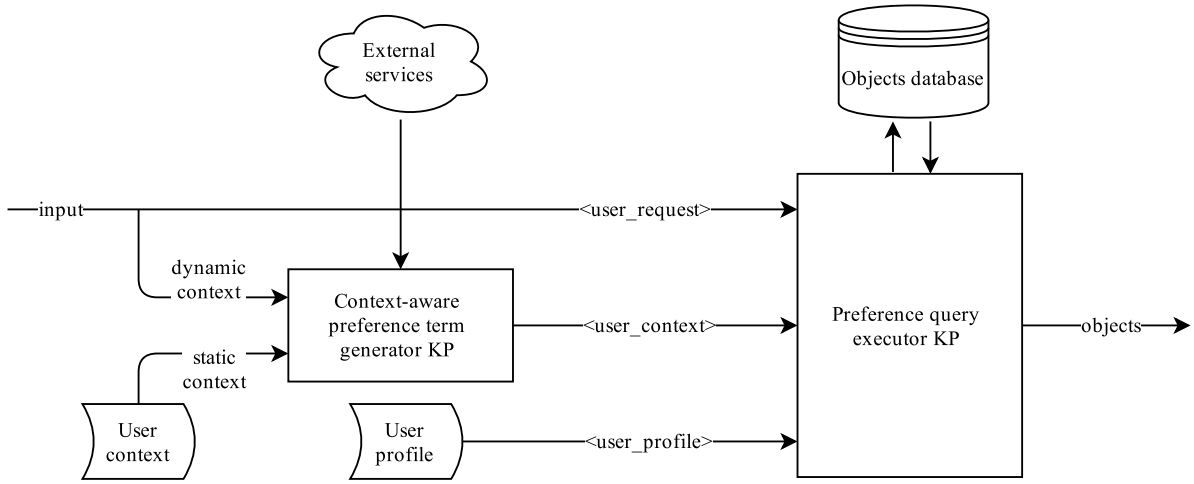

Fig. 1. The data flow of the framework

(CAPTGenerator). They take the user static and dynamic context properties and create preference terms from them according to the type of searched objects. In order to function a CAPTGenerator may rely on external services to get additional data that describes correlation between the user and searched objects.

The extraction of user context out of request and profile allows the whole system to be flexible. The request and the profile place restrictions to the queried objects directly, so the requesting service is only required to operate within the domain model of the data provider. On the other hand CAPT-Generators may operate with data from external domains, therefore making the system aware of the context. It also allows to extend system behavior without the need to modify behavior of the clients.

There are several KPs that perform particular tasks: *Client KP* performs request to the system and provides some services to the user, *CAPTGenerator KP* generates preference terms based on the dynamic and static user context, *Preference query executor (PQE) KP* gathers all preference terms related to the request and provides corresponding objects from the managed database. The rest of this section describes particular aspects of the framework operation.

C. Framework data structures

First part of the framework classes describe preference terms, their structure and relations are presented in Fig. 2 using Unified Modeling Language (UML) class diagram notation. The base class for all preference terms is an abstract class *PreferenceTerm*. All other classes of preference term elements that define *base preferences* and *complex preferences* are inherited from this class.

For all base preferences there is a basic abstract class *AttributePreferenceTerm* inherited from the *PreferenceTerm* class. It has an *attribute* field to indicate to which attribute of objects it is related. All classes for base preference constructors are inherited from the *AttributePreferenceTerm* class. For example, there are *BetweenPreferenceTerm*, *POSPreferenceTerm*, *LowestPreferenceTerm*, *LessThanPreferenceTerm* preference constructors classes. Each of them has fields that are necessary to perform their designation. The *BetweenPreferenceTerm* class has a *lower* and *higher* fields, the *POSPreferenceTerm* class has a *preferredSubset* field and etc.

In order to create a complex preferences there are *ParetoComposition* and *PrioritizedComposition* classes inherited from the *PreferenceTerm* class. Objects of the *ParetoComposition* class may contain a set of other *PreferenceTerm* objects with the use of *contains* property. Objects of the *PrioritizedComposition* class reference only two other *PreferenceTerm* objects – one of them in the *rightOperand*, another in the *leftOperand*.

The structure of classes that are used to represent client request to the system and response is presented in Fig. 3. The user context is modeled with the use of *UserContext* class that contains a set of properties. The context in real application should be modeled with the classes that describe corresponding aspects of the domain model, thought we intentionally simplify the representation of the context.

To store information about a user there is a *User* class that has user static context and user profile. The *UserProfile* class consists of a set of *UserProfileItem* objects. The *UserProfileItem* class contains a *PreferenceTerm* object. To identify to which type of database objects the *UserProfileItem* is related it has *objectType* field.

The static user context describes the user in some way and the user profile contains preferences that were directly stated by the user. Though the first one might be translated directly into concrete preferences by the Client KP this is not desirable, because this KP has limited knowledge of the domain. Another reason for separation is that context might be provided by other KPs and Client KP may not know the relation of these data with the request.

Client request for data is described by the *UserRequest* class. It consists of a user dynamic context and a request limits described by the object of *PreferenceTerm* object. To

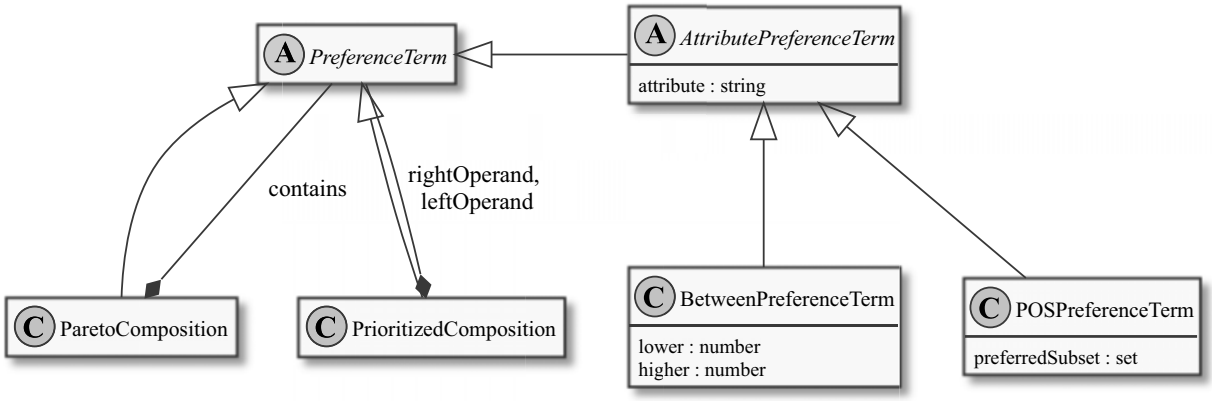

Fig. 2. Class diagram for preference terms classes

identify to which user request is related to it is linked with a corresponding *User* object. To identify to which type of database objects this request is related to it has *objectType* field. Also it has a *processed* field to identify when user request have been processed and result is ready. To store the result it contains a set of *ResultItem* objects.

The *ResultItem* class directly represents the objects that were found by the framework. The *ResultItem* classes reference a searched objects with the use of *consistsIn* property. The priority of extracted objects is indicated by the *position* field.

D. Client KP

The Client KP is not a singe KP, but a role inside the framework, that describes steps that are necessary to retrieve context-bound object list. In order to achieve this task Client KP should provide a description of the user, one's preferences and perform an object retrieval request. This role could be implemented by the KP that interacts directly with the user of the system.

The Client KP knows the domain model of the concrete system and provides user with appropriate tools to formulate requirements for the system. For example, if a requested object has a price attribute, then the KP may provide input fields to specify minimal and maximum desired values. Then they will be used to form corresponding preference terms for the request.

Initialization and subscriptions:

When the KP connects to the smart space it either adds or locates an objects that describe the user: *User*, *UserContext* and *UserProfile*. KP must check that the object of the class *UserProfileItem* that describes preferences for particular object type is present and contains relevant data. Client KP may update these data if the user changes ones opinion.

When a user requests the objects from the system, the KP forms a request by adding a *UserRequest*, *UserContext* and *PreferenceTerm* objects to the SIB. Then it subscribes to the *processed* property of the added *UserRequest* object to detect the moment when request is processed. The subscription pattern is the following: (*UserRequest ID*, 'pqe:processed', 'true'). The subject holds the identifier of the added object.

Notification handler:

When the request is processed the KP queries results from the SIB and displays it to the user in some way. Then it removes subscription to the *processed* property and deletes all data that correspond to this request, i.e. the object *UserRequest* and all other object that directly relate to it except the *User* object.

Finalization:

When the KP stops it's work, it removes the base user preferences towards this kind of objects (the *UserProfileItem* that describes them). If there is no other preferences left, then Client KP removes other parts of user description from the SIB including the *User* object and the corresponding *UserContext* object.

E. CAPTGenerator KP

The purpose of these KPs is to convert dynamic and static user context into preference terms that affect the queries for a certain type of objects. The concrete system may include several preference generators even for single object type. All generated preferences have same importance between one another and concatenated with the Pareto preference.

Generators could be specialized and process only a subset of data that is available during the query execution. Generators can be shared between several object types if they share field types and their designation. There is no guidelines on how small or big generators can be, it is up to developer to decide on the modularity of preference generators.

Initialization and subscriptions:

After the connection to the smart space CAPTGenerator KP adds a *ContextAwareGenerator* object to the shared storage. It indicates the presence of the particular generator and specifies the type of object that it is capable create preferences for. If generator can formulate preferences for a set of objects, then for each of those objects generator must add an object of *ContextAwareGenerator* class.

This KP tracks the addition of particular *UserRequest* objects to the SIB. This can not be directly achieved through triple-pattern subscription, so KP must track addition of

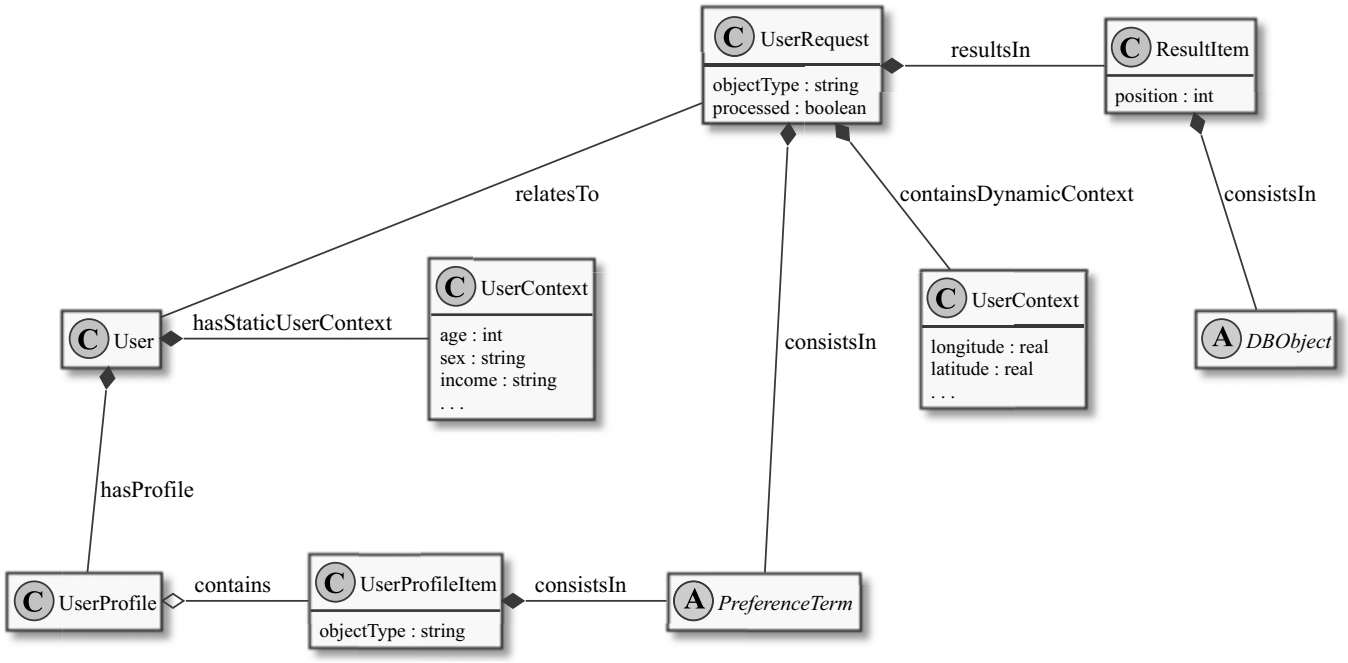

Fig. 3. Class diagram for request presentation and processing

all requests and filter them manually. Therefore, KP subscribes to the following triple pattern (ANY, 'rdfs:class', 'pqe:UserRequest') that allows to get an id of the added *UserRequest* object.

Notification handler:

When the SIB notifies CAPTGenerator KP, the KP queries the type of the object that was requested by the client. If it does not match with the one processed by the generator, then this notification is ignored. Otherwise if it matches the KP specifications, KP reads dynamic context associated with the request and static context bound to the user and generates preference terms out of these data. It may access external sources in order to perform this operation.

When the CAPTGenerator KP prepares the preference term it adds a *ProcessedRequest* object into the SIB. The presence of the object identifies that the processing is over and provides or do not results of this processing in *PreferenceTerm* object referenced with the *resultsIn* property.

Finalization:

When the CAPTGenerator KP disconnects it removes corresponding *ContextAwareGenerator* object from the shared storage. The generator does not need to remove products of it's work, because they relate to the client requests and the client removes them after it finishes processing returned data.

F. PQE KP

This KP uses the user input, the both of user's contexts and user's profile preference terms to provide relevant objects from the database. In order to do so PQE must keep a track of CAPTGenerator KPs for each object type it provides and data requests form Client KPs.

Initialization and subscriptions:

The PQE KP does not provide it's description into the smart space, but only tracks changes in it. Firstly, it subscribes for addition and removal of objects of the *ContextAwareGenerator* class. It can be achieved with the subscription to the following template (ANY, 'rdfs:class', ContextAwareGenerator).

Secondly, the PQE KP subscribes to addition of objects of the *UserRequest* class and *ProcessedRequest* class. The first is tracked with the subscription (ANY, 'rdfs:class', UserRequest). The latter is tracked with the subscription to (ANY, 'pqe:generates', ANY). The last subscription allows to detect the CAPTGenerator KP that added this preference.

Notification handler:

When the PQE KP receives a notification about addition of *ContextAwareGenerator* class object it queries full description of the generator and adds it to the list that correspond to particular object type. When the removal notification is received then the description is removed from the list.

When the PQE KP gets notification from one of two last subscriptions it firstly check the number of *ProcessedRequest* class objects related to the particular request object of the *UserRequest* class. If this number is less than the number of generators that correspond to this type of queried objects then this notification is neglected. Otherwise it combines the user input, preferences generated from the context and profile preference terms together by previously stated prioritization order and executes this query on the object database.

When the database query has finished, the PQE KP adds those objects to the corresponding *UserRequest* object in the form of *ResultItem* objects. Also KP assigns "true" to the value of the *processed* property of the *UserRequest* object. As the

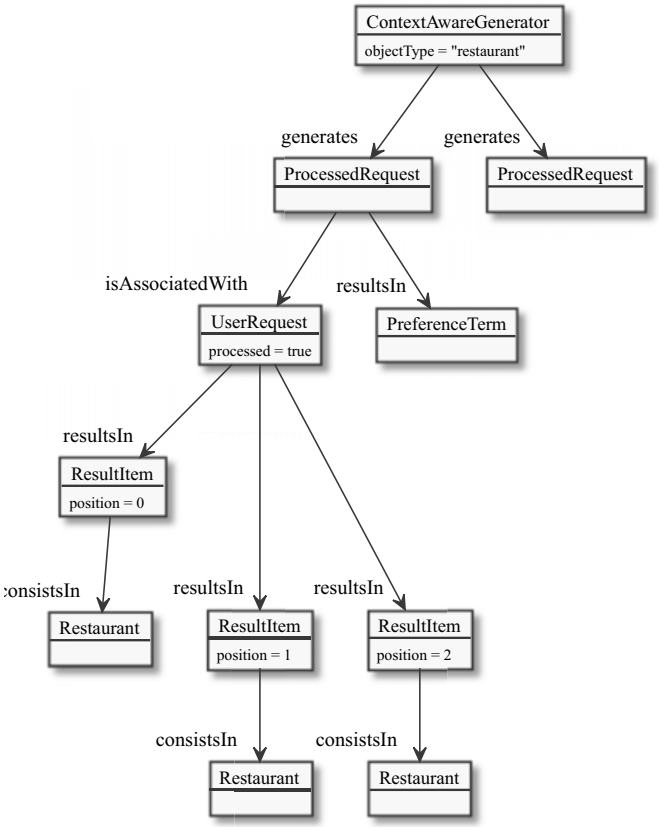

Fig. 4. Data structures of executive mechanisms

last step the KP removes data provided by the generators to this query from the SIB.

The data structures produced by CAPTGenerator KP and PQE KP are illustrated in Fig. 4

V. Case study: context-aware restaurant data retrieval

In this section we consider an example of how the proposed framework can be used to build a location-based service that provides information about restaurants. This service allows a user to search for nearby restaurants that are most relevant according to his/her request, context, and profile.

In order to create this type of service on top of the framework developers should implement several CAPTGenerator KPs and a PQE KP. The CAPTGenerator KPs generate a series of preferences for a "restaurant" domain taking into account the location of the user, and the PQE KP has access to database that describe objects of interest. The end-user should interact with the system with the use of mobile application that acts as the Client KP. The following scenario shows how these components interact with each other to provide a satisfactory result to the user.

Suppose that a user wants to find a restaurant nearby his current location to have a meal. The user is male and vegetarian. These data are entered by the user into the client application on first start, they describe static context. Also the user specifies that he does not want to spend more than 40 dollars for the lunch and likes Italian cuisine. These rules form the user preferences for the "restaurant" domain.

The Client KP processes these data and describes them inside the shared storage in the form of data structures described in Section IV-C. The diagram of objects added to the SIB is presented in Fig. 5. The KP adds the object of the *User* class, places context data (sex, food choice) into the object of the *UserContext* class. The preferences are represented by the object of the *UserProfileItem* class with the *objectType* property holding the value of "restaurant". This object points to object of the *ParetoComposition* class that combines the object of the *PosPreferenceTerm* for specifying the cuisine with the object of the *LessThanPreferenceTerm* class for the lunch price. The formal representation of the base preferences is the following:

$$\text{<user\_profile>} := \text{POS(cuisine, 'italian')} \otimes \text{LESS\_THAN(lunch\_price, 40)}$$

The user wants to find a restaurant with the rating between 2 and 4 starts with the cheapest menu and specifies that price requirement is the most important at the moment. The Client KP translates this request into a set of preferences and adds the object of the *UserRequest* class to the SIB. This object refers to the previously added object of the *User* class. Also this object references new object of the *UserContext* that contains current location of the user and represents dynamic context. Generated preferences are presented by the object of the *PrioritizedComposition* class that places the object of the *LowestPreferenceTerm* class for the price to be more important than the overall rating of the restaurant that is represented by the object of the *BetweenPreferenceTerm* class. The formal representation of the client query is the following:

$$\text{<client\_request>} := \text{BETWEEN(rating, 2, 4)} \,\&\, \text{LOWEST(lunch\_price)}$$

When the *UserRequest* object is added into the shared storage the CAPTGenerator KPs start to process the context of the request. When they finish, each of them places data structures to represent the preferences query in accordance with the format described in Section IV-E. The first generator is capable to convert user's food preferences into the availability of special kinds of food in the menu. The formal representation of preferences generated out of the static user context can be represented as the following:

$$\text{<user\_context>} := \text{POS(vegetarian\_menu, true)}$$

Another CAPTGenerator KP is capable to create preference term based on the location of the user provided inside the dynamic context. This KP is not tied to the restaurant domain, but only to the location part of the service. It generates a preference term to search for object near by the specified location. The formal representation of preference is the following:

$$\text{<user\_context>} := \text{NEARBY(latitude, longitude)}$$

When these generators have formulated preferences and added corresponding objects of the *ProcessedRequest* and *PreferenceTerm* classes to the SIB the PQE KP starts to query data from the database. It combines preference terms received from the client request, preferences generated from the context and profile preferences and formulates the following result preference term:

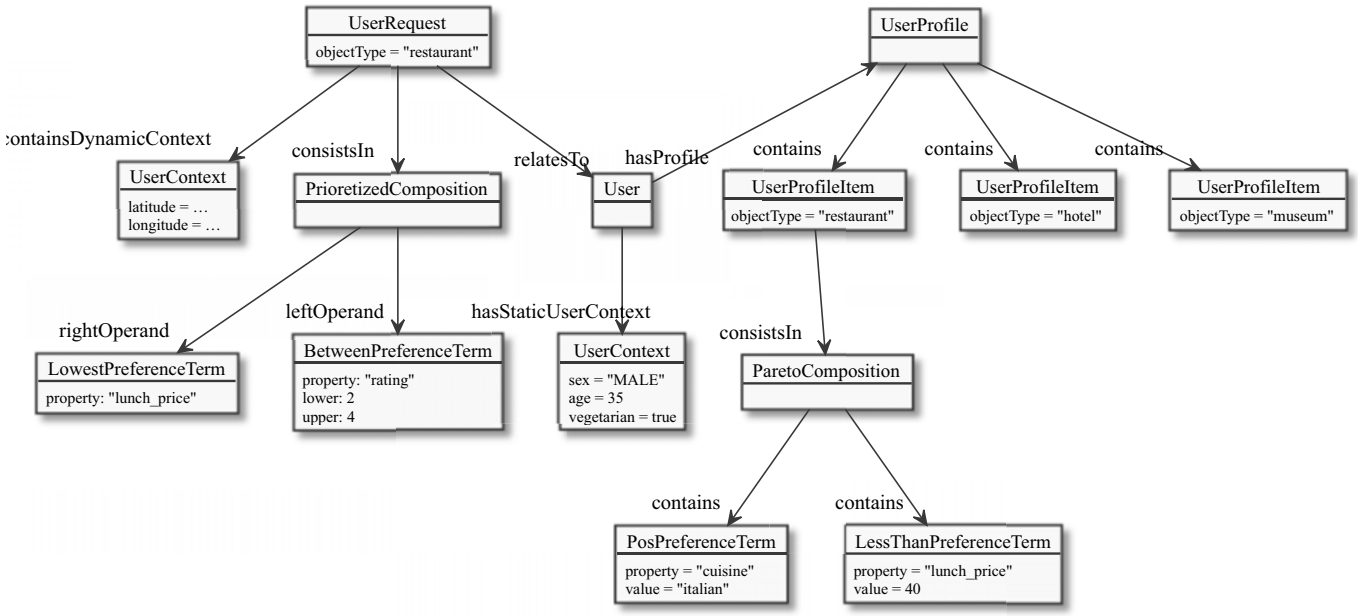

Fig. 5. Object diagram for restaurant finding example

<preference_term> := BETWEEN(rating, 2, 4) & LOWEST(lunch_price) & NEARBY(latitude, longitude) ⊗ POS(vegetarian_menu, true) & POS(cuisine, 'italian') ⊗ LESS_THAN(lunch_price, 40)

Thereafter, the PQE KP executes the query on top of the database, takes the resulting list of objects and binds it to the object of the *UserRequest* class. Also it sets "true" as the value of the *processed* property of that object.

When the Client KP receives notification about the last change it determines that the processing of the request is over and the result is ready. Then the KP queries resulted data from the shared storage and displays it to the user. When the query is no longer needed by the user, the Client KP removes corresponding data from the SIB.

VI. Conclusion

In the paper we presented a conceptual framework for development of context-aware location-based services for Smart-M3 platform. It is based on a similar framework described in [6] with a proper adaptation for architecture of services on Smart-M3 platform.

The main components of the framework include context-aware preference term generators (CAPTGenerators) that translate context information into context-aware preference terms and preference query executor (PQE) that combines all preference terms and conducts their execution using Preference SQL JDBC driver.

The paper describes behavior of all the framework components. Evaluation of the proposed approach is made using the case study of context-aware restaurant data retrieval.

The directions for the future work include the development of architectural approach to define non-functional preferences, i.e., support several providers that can rank objects based on some compound logic. This feature would allow to extract GIS components into a separate entity out of the execution core, add alternative rank provider, and therefore increase flexibility of the framework.

Acknowledgment

This research is financially supported by the Ministry of Education and Science of the Russian Federation within project #14.574.21.0060 (RFMEFI57414X0060) of Federal Target Program "Research and development on priority directions of scientific-technological complex of Russia for 2014–2020".

References

[1] A. Zipf *et al.*, "Location-based services," in *Springer Handbook of Geographic Information*. Springer, 2012, pp. 417–421.

[2] A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[3] E. Kaasinen, "User needs for location-aware mobile services," *Personal and ubiquitous computing*, vol. 7, no. 1, pp. 70–79, 2003.

[4] B. Sadoun and O. Al-Bayari, "LBS and GIS technology combination and applications," in *International Conference on Computer Systems and Applications*. IEEE, 2007, pp. 578–583.

[5] J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, "Smart-M3 information sharing platform," in *IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2010, pp. 1041–1046.

[6] P. Roocks, M. Endres, H. Alfons, W. Kießling, and S. Mandl, "Design and implementation of a framework for context-aware preference queries," *Journal of Computing Science and Engineering*, vol. 6, no. 4, pp. 243–256, 2012.

[7] J. J. Levandoski, M. E. Khalefa, and M. F. Mokbel, "An overview of the CareDB context and preference-aware database system," *IEEE Data Eng. Bull.*, vol. 34, no. 2, pp. 41–46, 2011.

[8] R. Yus, E. Mena, S. Ilarri, and A. Illarramendi, "SHERLOCK: Semantic management of location-based services in wireless environments," *Pervasive and Mobile Computing*, vol. 15, pp. 87–99, 2014.

[9] A. Smirnov, A. Kashevnik, N. Shilov, and N. Teslya, "Context-based access control model for smart space," in *5th International Conference on Cyber Conflict (CyCon)*. IEEE, 2013, pp. 1–15.

[10] M. Saleemi, N. Rodriguez, J. Lilius, and I. Porres, "A framework for context-aware applications for smart spaces," *Smart Spaces and Next Generation Wired/Wireless Networking*, pp. 14–25, 2011.

[11] D. Korzun, I. Galov, and S. Balandin, "Development of smart room services on top of Smart-M3," in *Proceeding of the 14th Conference of FRUCT Association*. IEEE, 2013, pp. 37–44.

[12] K. Krinkin and K. Yudenok, "Geo-coding in smart environment: Integration principles of Smart-M3 and Geo2Tag," in *Internet of Things, Smart Spaces, and Next Generation Networking*. Springer, 2013, pp. 107–116.

[13] A. Smirnov, A. Kashevnik, N. Shilov, H. Paloheimo, H. Waris, and S. Balandin, "Development of broker logic for ridesharing system on top of Smart-M3," in *Baltic Congress on Future Internet Communications (BCFIC Riga)*, Riga, Latvia, 2011, pp. 190–197.

[14] A. Smirnov, A. Kashevnik, N. Shilov, N. Teslya, and A. Shabaev, "Mobile application for guiding tourist activities: Tourist assistant—TAIS," in *Proceedings of the 16th Conference of Open Innovations Association FRUCT*. IEEE, 2014, pp. 95–100.

[15] W. Kießling, "Preference queries with SV-semantics." in *COMAD*, vol. 5, 2005, pp. 15–26.

[16] W. Kießling, M. Endres, and F. Wenzel, "The Preference SQL system—an overview." *IEEE Data Eng. Bull.*, vol. 34, no. 2, pp. 11–18, 2011.

[17] F. Wenzel, D. Köppl, and W. Kießling, "Interactive toolbox for spatial-textual preference queries," in *Advances in Spatial and Temporal Databases*. Springer, 2013, pp. 462–466.

[18] M. Schneider, J. Carroll, J. Herman, and P. Patel-Schneider. OWL 2 web ontology language RDF-based semantics (second edition). [Online]. Available: http://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/