‘What Affects Me?’ A Smart Public Alert System based on Stream Reasoning

Snehasis Banerjee  
Tata Consultancy Services  
TCS Innovation Labs  
Kolkata, India, 700091  
snehasis.banerjee@tcs.com  

Debnath Mukherjee  
Tata Consultancy Services  
TCS Innovation Labs  
Kolkata, India, 700091  
debnath.mukherjee@tcs.com  

Prateep Misra  
Tata Consultancy Services  
TCS Innovation Labs  
Kolkata, India, 700091  
prateep.misra@tcs.com  

ABSTRACT
Public alert services is gradually becoming popular in smart cities because this enhances the awareness of the citizen about activities within the city. Such a service also ensures the safety and security of the citizens. However the state of the art lacks in providing real-time alerts in a personalized, context-aware fashion utilizing the combined knowledge about the city, its events and its citizens. In this paper, a solution architecture is presented that uses stream reasoning as its backbone which suits the domain of a public alert system very well. The stream reasoner uses rule-based reasoning and queries. The rules are designed as atomic concepts. A fully functional prototype of the proposed system was developed and tested on data of a smart city. The experimental results support that the proposed methodology is very effective.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous

General Terms
Design

Keywords
personalized alerts, context awareness, stream reasoning

1. INTRODUCTION
With the rapid advancement of technology and government backing, citizens of modern cities are getting access to smart services that enhance their citizenship experience. Modern cities providing such services are called smart cities [7]. Public alerting system is one such important smart city service. A public alert system conveys alerts regarding various events such as traffic disruption, crime, hazard and emergency situations in a city to its citizens.

Public Alert systems have been in existence for quite long. Earlier the alerts were conveyed through radio and TV, but now with the emergence of Internet and Telecommunication email alerts, SMS and real-time posts on social platform have become the preferred medium of communication.

To ensure that no city based incident is ever missed, the system needs to sense every change in environment and conditions in the city. This is made possible as modern cities are gradually getting equipped with various types of sensors that can sense events as and when it happens.

Physical sensors like thermometer and hygrometer give readings based on which a concerned authority can raise weather alerts of high humidity and heat. The concerned authority can also raise alerts about traffic disruptions and crime by analysing live feeds from video surveillance equipment fitted at various locations in a city or on notification by traffic police authority. Another interesting way to sense events is crowd sensing [2]. In crowd sensing, the data is posted by citizens on social platform based on their personal observation or experience; and this data can form the basis of alerts. There are also other channels based on which alerts are forwarded by the concerned authority to the citizens.

In some cities only emergency alerts like flood or high tide that affect a large population is forwarded. Some other modern cities in the world has facility to provide more specific alerts and information services on a subscription basis to its citizens. As the number of daily events is large, a citizen is flooded with alerts that are irrelevant to him or her. Some city alert services provide a choice to the subscriber (citizen) to statically state a set of location preferences on a web portal. However that facility is not good enough as the citizen’s location in a city is not static, but dynamic; so irrelevant alerts will be delivered and some important alerts will be missed out. For example if a subscriber (who is a citizen of New York) has stated that he stays in Columbus Avenue, but currently he is at Wall Street which is mid-way to office, so an alert about a traffic jam at Columbus Avenue is useless to him at that moment. Similarly if a traffic incident happens in Fulton Street which is his next route segment on way, he will not be alerted as Fulton Street as a location of interest to the user is not stated in his preference. Apart from events that will affect the citizen directly he may be interested in events that are affecting or will affect their near and dear ones. Also only location based preferences are not good enough to filter alerts. User’s general preferences (like preference for weather alerts only) and personal profile (like user’s medical condition) information are also important to give more specific alerts. Hence there exists a need for a real-time system that is intelligent enough to filter out the relevant alerts.
Problem Statement: As there are many events happening in a city frequently, it is difficult to notify the citizens about the relevant events that are currently affecting them or will affect them in future. Such a selective notification will enhance the citizen’s ubiquitous experience.

Figure 1: Problems of existing public alert systems

One way to achieve such intelligence is to use the citizen’s preferences, social web presence and demographic information to ensure that a citizen receives personalized alerts. Another way is to make the system context-aware. According to [3], a context-aware application provides relevant services to an entity based on the information that characterizes the situation of the entity (here citizen is the entity). Examples of context include location of the entity, time of day, weather conditions, companion. Personalization and context-awareness can be further enhanced if the information gathered from these approaches can be extended. This extension of freshly gathered information into new information using existing knowledge is called reasoning. As in the system under consideration, real-time streams of event and context need to be reasoned upon to derive richer knowledge, stream reasoning is the way to go. A stream reasoning system [17] does reasoning on the fly on streams of knowledge combined with background knowledge to produce meaningful entailments. However, the lifetime of the entailments is short-lived as the facts causing the entailment generation are themselves short-lived due to the dynamic nature of the real-time streams of knowledge. User managed window [13] is a good way used by a stream reasoner to ensure that no obsolete knowledge exists in the system. The aforementioned techniques are discussed in more details in section 4 and 5.

To address the problem discussed earlier, a near-real-time system has been designed that is
1) personalized: alerts are based on citizen’s preferences, demographics and information in the social web
2) context-aware: alerts are forwarded to the citizen depending on the citizen’s current and future context
3) knowledge-based: reasoning is done on the fly to infer dynamically changing knowledge that aids the system to decide whether to forward an alert to a particular citizen or not.

The main contributions of our work are as follows:
1) a solution architecture and methodology based on stream reasoning for public alert domain has been designed
2) a fully functional prototype of the solution has been developed and tested on smart city data
3) investigation on applicability of atomic concept-based rules in stream reasoning scenario is carried out.

Thus compared to the state-of-the-art, the solution brings together context-awareness, personalization, near-real-time and knowledge-based aspects to the public alert domain to achieve the goal of providing a superior service to the citizens of smart cities.

The remainder of this paper is outlined as follows: Section 2 discusses the prior works relevant to aforementioned problem. Section 3 examines the problem in detail and comes up with user requirements and motivating examples. Section 4 discusses different approaches to the solution before selecting stream reasoning as a solution. Section 5 presents the design of the solution to the problem. Section 6 illustrates a working prototype of the solution. Section 7 does a thorough analysis of results obtained through experiments. Section 8 summarizes the work while section 9 portrays future research directions.

2. RELATED WORK

In this section, some existing public alert systems are looked into before discussing about the relevant literature. It is found that a number of public alert services exist because of the importance of such a service. PLAN1 is a new public safety system introduced by US Government where wireless carriers push emergency safety alerts from cell towers to mobile phones in the affected area. StormWatch+2 is an alerting system that pushes weather alerts to subscribers based on subscriber’s location. Recently Google has introduced a Public Alert Service3 integrated into Google Maps that shows emergency alerts about weather, flood and earthquake based on feeds from government agencies.

One of the earlier investigations carried out in the public alert domain is by Mileti et al. [10] that reviewed more than 200 emergency public warning systems to bring in a social science perspective into the subject. Some of the key findings were a) whether a citizen heeds a warning depends on various factors like accuracy, understandability and frequency of message b) response taken by a citizen to an alert depends on various factors like demographics and profile c) the medium of choice to communicate alerts should based on the warning type and severity. What is interesting from our paper’s perspective is that if a citizen receives too many alerts and most of them are irrelevant, then the citizens tend to neglect the alerts conveyed by the alert system. It was also found that an alert system needs to know the user well. These issues have been addressed in our work. Botterell et al. [1] has brought into light the importance of open standards in alerting system. Also the need for the public alert system to be bi-directional (not just governments sending alerts to citizens) is described. Our proposed system also monitors social web feeds and aggregates information from various sources (not only government) to provide alerts that might have been missed if dependence was on a single centralized source. Our proposed system works with knowledge triples namely RDF4 that follow the semantic web standard. A wrapper layer converts the standard forms of alerts (CAP5 and RSS) to their corresponding RDF representation. Sedighi et al. [16] has proposed a public alert system architec-

1http://transition.fcc.gov/pshs/services/plan.html
2http://www.stormwatchplus.com/
3http://support.google.com/publicalerts
4http://www.w3.org/RDF
5http://www.incident.com/cap/what-why-how.html
ture and has conducted some field trials and experiments. Most of the challenges they have pointed out (like targeting citizens based on geographic location) have been addressed in our work. Hu et al [8] has described a rule-based architecture for provision of information relevant to user’s activities in a city. The work also introduces an ontology-based context model to characterize the possible situations of a citizen in a city. Montanari et al. [11] proposes an architecture for a policy driven warning system, where the alerts are disseminated based on some pre-set policies. The alert topics are represented as triples while the crisis policies are maintained as rules which are fired when corresponding conditions are met (like if an alert is severe, activate all mediums to convey the alert such as radio, TV and city alarms). The main differentiators of our work from the prior works discussed above is application of stream reasoning based on concept-based rules and overall solution architecture for the public alert scenario. Our proposed system gathers information from different sources and combines the knowledge of the city, its events and its citizens to ensure delivery of relevant alerts.

3. MOTIVATION FOR SOLUTION

What a citizen might actually want from a public alert system is discussed in this section. Some motivating examples are also illustrated that show both the intelligence requirements and the importance of such a system.

3.1 Requirement Analysis

A summary of main requirements of a citizen w.r.t. the alerts that he or she will like to receive is listed below:

- **High Relevance**: A citizen should be provided only relevant public alerts. For example, an alert about a traffic jam at a place in Japan is of no use to a person who has gone to New York to attend a conference.

- **Minimum Latency**: A citizen should be alerted immediately. For example, if the citizen gets an alert about some traffic jam when he has already crossed the affected location (and has got late to office) is both useless and frustrating.

- **No Misses**: No relevant alerts should be missed. For example, shooting event was happening in a citizen’s current path to office, but somehow the system marked this alert as irrelevant. This breaches the safety and security of the user. Such a situation can occur either if a probabilistic approach is taken or if knowledge about the user’s future context is not well defined or unknown.

3.2 An Exemplary Scenario

Bill is a citizen of New York. Bill works as an accountant at a company named Nanosoft, which is located at ABC Building. He lives with his wife Marie and son Steve at W 82 Street beside Columbus Avenue. Marie is a professor at Polytechnic University of New York at Long Island center on Maxess Road. Steve studies at at Edward Bleecker School, located beside Hart Playground at 27th Avenue. His wife likes driving the car, which is used to drop Bill and Steve at school. Montanari et al. [11] proposes an architecture for provision of information relevant to user’s activities in a city equipped with event sensors. The work also introduces an ontology-based context model to characterize the possible situations of a citizen in a city.

There are many other irrelevant events happening in the city about which Bill is never alerted in a real-time fashion.

3.3 Specific Examples

In this section a few use cases are presented corresponding to some events. We list down each event and then list down conditions in which the event is relevant to a citizen and when it is not. We define some generic entities:

- **C**: a smart city equipped with event sensors
- **L_i**: some location i in C
- **UE**: a citywide event that affects all X, like hurricane
- **CE**: a citywide event that affect X based on context
- **PE**: a citywide event that affect X based on X’s profile
- **LE**: a city location specific event, ex. traffic disruption

We list down each event and then list down conditions in which the event is relevant to X:

**Case-1**

Event : UE\_1 (say a storm) will happen.
R(UE\_1, X) : X is currently located far from L\_1
I(UE\_1, X) : X is currently on or near L\_1

**Case-2**

Event : CE\_1 (say rain) will happen
R(CE\_1, X) : X is currently on the road or about to go out
I(CE\_1, X) : X is inside a building and has no plans to go out

**Case-3**

Event : LE\_1 has happened (say shooting is going on at L\_1)
R(LE\_1, X) : X is currently located far from L\_1
I(LE\_1, X) : X is currently located near L\_1
Case-4
Event : Some PE₃ event is forecast (say extreme heat alert)
R(PE₃,X) :- X has hyperthermia allergic condition
I(PE₃,X) :- X does not have heat related disease

4. APPROACH TO SOLUTION
In this section possible solution approaches are explored. Before that we list the resources that may be used to reach a solution. There are broadly two types of data that may be available in a smart city:
a) Static Knowledge such as data about routes, roads, locations, places of interest, citizens’ profile.
b) Dynamic Knowledge such as feeds about various events, dynamic context data of citizens.

From discussions in earlier sections it has been identified that the public alert system needs to be context-aware, personalized, knowledge based and near real-time. Each of this features are individually discussed.

4.1 Personalization
A personalized application delivers alerts to targeted users based on their profile information. Profile information can be either manually stated by the user or automatically mined from the social web. Profile information include their age, gender, places of interest, habits, profession, relatives and common activities they involve in. An application can also be made personalized based on a model learnt over time based on the citizen’s behaviour and activity using machine learning techniques. Most literature such as [12] has found that combining both user preferences and user behaviour yields the best personalization experience.

4.2 Context-awareness
Context-aware applications use the current situation of a user to provide suitable services. In the last decade, a lot of work has been done to enable context-awareness in different applications. Context relevant to a user include the time of day, the location, companion, current activity and environmental conditions. The context information is needed to be sensed by dedicated sensors (ex. location can be sensed by GPS device) before any intelligent processing can be done.

4.3 Knowledge based System
A system that provides intelligent decisions or justification based on some specific domain is a knowledge based system. Usually rules are applied on existing knowledge to infer new knowledge. Ontology is a good way to represent a set of concepts in a domain and their relationships. Reasoning can be applied on facts to get entailments based on the defined ontology. Facts are triples of the form [subject : predicate : object]. Both personalization and context-awareness can be modelled based on ontology and rules as described in [5] and [18]. This will enable the system to make better decisions based on the extended knowledge.

4.4 Real-time System
A real-time system needs to guarantee response within strict time constraints. In the scenario under discussion, there needs to be some way to process the streams of context and events very fast. Current complex event processing systems does on-the-fly analysis of streams but is incapable of doing on-the-fly reasoning. That shortcoming is fulfilled by stream reasoning which is a near-real-time technique of reasoning on static (or slowly changing) background knowledge and time varying knowledge derived from sensors and other sources.

A modern city contains huge volume of static data and real-time sensor data and feeds. Hence, the problem discussed above can be solved by a stream reasoner and some enabler systems.

4.5 Stream Reasoning as a solution
Let us look at a stream reasoning application that sends alerts about traffic accidents to its citizens. Let $K₁$, $K₂$, $K₃$, $K₄$ and $K₅$ be some statements:-

$K₁$: Citizens $C₁$ and $C₂$ are affected by $T$
$K₂$: There is traffic accident $T$ at street $X$
$K₃$: Street $X$ has latitude $P$ and longitude $Q$
$K₄$: $C₁$ is currently on way towards street $X$
$K₅$: Citizen $C₂$ is a close relative of $C₁$

Here $K₂$ is knowledge about an event that will become false after a time period. $K₄$ is the knowledge about the dynamic context of the citizen. $K₅$ is knowledge gathered from the citizen’s social network. $K₂$ is a static geographical knowledge. It can be seen that $K₃$ and $K₄$ form the background knowledge while $K₂$ and $K₅$ form the dynamic knowledge. $K₁$ is not an obvious knowledge but can be derived by combining $K₂$, $K₃$, $K₄$ and $K₅$. This linking of both static and dynamic knowledge to entail new knowledge is done by stream reasoning based on some rules. Here the rule to derive $K₁$ will be of the form $[K₂, K₃, K₄, K₅ \rightarrow K₁]$. If any of $K₂$ or $K₄$ change, $K₁$ will become invalid. So the system needs to update the truth values of derived facts on a near real-time basis to keep the knowledge up to date.

So a stream reasoning platform uses two kinds of knowledge: static and dynamic, to derive new knowledge that is useful for the application. There needs to be some way to access the knowledge and that is done by querying the combined knowledge to find if a knowledge pattern exists. The standard way to represent knowledge in stream reasoning is RDF which is a triple format. This follows the vision of semantic web, where the end-users will be machines. SPARQL is the standard query language used to find patterns in RDF graphs. Thus it is seen that using stream reasoning, it is possible to bring context-awareness, personalization, near-real-time and knowledge based aspects to the solution.

5. SOLUTION DESCRIPTION
In this section detailed description of the solution is presented. The technical challenges and research issues are also discussed.

5.1 Solution Architecture
We first discuss the process flow to get a general feeling about the how the system works before going into the details about the individual modules of the system.

5.1.1 Process Flow
Citizens will need to subscribe to a service that will provide relevant public alerts to them. The citizens are needed.
to provide their personal profile data. The profile data can be collected through mobile application or web portal. The same can be fetched from social web also. As in any knowledge based system, the more useful information or knowledge the system has, the more intelligent the system will be, provided there is some means to use and understand the knowledge. Profile will include citizen’s preferences like which alert types is the citizen interested in (ex. a news reporter may be interested in crimes rather than traffic jams), demographics and social information like family and (close) relatives. This will also include the citizen’s activity in the city, like locations often visited. The context data about citizen like current GPS location will also be needed by the system to provide the alert service. GPS information can be posted by a mobile application to the system. The system will have access to the city’s spatial database. The system will gather real-time event feeds in different formats from various sources like agencies and web. The relevant feeds will be filtered by the system and forwarded to the user’s device.

5.1.2 Module Description
The individual modules shown in Fig 3 is described here.

User Interface: The user (in this case a citizen) needs to enter profile data here and optionally specify his or her social web presence that will be used by the application. Alerts will also be displayed on this medium.

App: The purpose of the application is to convey the results to the end users. This also form a medium by which the profile data is stored and updated in the background knowledge module. Database DB1 represents the social web data that can be extracted using social web mining techniques as discussed in [15]. Data from DB1 is processed by Profile Updater module to extract relevant data. The application developer has the responsibility to provide the queries and rules that will run in the stream reasoner. The Query Listener listens for query results coming from the stream reasoner. The Alert Processor processes the results and sends the alerts to the intended users. In between, the Alert Translator translates the alerts to the language of the specific user’s choice and formats the alerts customized to user’s intended UI and medium of communication.

Sensing: This module has the responsibility to sense the environment in a city and detect events. This task is usually carried out by sensing and posting services of dedicated agencies (both government and private). Events in social web are also sensed by keeping real-time track of human-generated content.

Monitoring System: This module monitors the heterogeneous sources of sensed data (like social web posts and agency alert feeds) and has built-in logic to differentiate events and context data. An important activity carried out here is to forward only unique data, it does a duplicate check by keeping track of recent data that has been processed.

Knowledge Converter: This module converts raw text to structured knowledge format that is understood by a stream reasoner (like RDF which is a standard way to represent a triple T). For structured data that follow some open standards, the conversion to knowledge is easy. For semi-structured data, this can be done by relevant keyword based match. The keywords can be manually tagged or learned over time by analysing past data. For unstructured data some training is done at first by tagging what the text conveys. Models can be learnt by applying machine learning to classify the texts. One way to understand the unstructured data is to use natural language processing and information extraction techniques. Knowledge-pattern based sentence conversion to knowledge is another way as proposed by [9]. It is to be noted that the stream reasoner under discussion takes dynamic data in the form of knowledge packets (KP), an important concept introduced in [13]. KP is a set of triples that carry some knowledge in unison. KP as a communication medium disallows any partial entry of knowledge into the stream reasoner.

Context Extractor: This module collects current context from the user’s device (like GPS) and Monitoring system. Mapping relationship of sensed data to context is stored in DB2, ex. raw GPS locations are mapped to some landmark
locations, raw time is mapped to some daypart and temperature is mapped to hotness or coldness. The future context like planned route of a user or weather forecasts are also gathered. The data is converted into knowledge packets before being forwarded to the stream reasoner.

**Background Knowledge:** The knowledge about the city like streets, points of interest and its nearby locations along with static data about users is stored here in a structured format. Rules are applied on the background knowledge to extend the knowledge. Then only that portion of knowledge that is needed by the application, is loaded. In some cases the base background knowledge will not be loaded but the derived knowledge will be. For example the information that two locations lies in same neighbourhood infer that they are mutually near each other. An application may only need the knowledge about nearness of those two locations, not their neighbourhood names. This takes load off from the stream reasoner that now can focus more on processing the real-time knowledge streams of context and events.

**Stream Reasoner:** The stream reasoner (SR) used in this application is the one introduced in [13]. The SR supports an API by which client applications can a) send events b) register listeners for specific continuous queries. Events are sent as sets of triples known as knowledge packets (KP). The Event Manager receives the KP and schedules it for query execution by the Query Executor. Before execution, the triples are added to application specific working memory also called Memory Area. On addition of triples, reasoners act on the new knowledge producing further knowledge (entailments). Currently the SR supports rule-based reasoners and provision for adding other reasoners are available. Reasoning is done before query execution. The results from Query Executor is passed to the Result Processor module which passes the results to the listeners registered by the SR clients. The SR supports user-managed windows introduced in [13] which allows client applications to control when an event should be deleted. Context is also packaged as an event (KP) and sent to the SR.

**5.2 Technical details**
Stream reasoning systems run continuous queries and reasoning on the knowledge to accomplish its task. The stream reasoner in our solution uses rule-based reasoners and queries. The application will supply the queries to run and rules to fire in the stream reasoner. The application will listen on those registered continuous queries over the combined knowledge to fetch results, if any.

Basically the whole logic of reasoning can be embedded in a SPARQL query. For example if the following SPARQL query:

```sparql
select ?location2 where
{ ?person s:isLocatedAt ?location1 .
  ?location2 s:isNear ?location1 }
```

is run on a knowledge store of geographical information, it will return places located near the person’s current location. This approach is referred in the paper as query-based approach. The logic to get the locations near a person’s current location can be written in the form of rules as: (here we use the syntax of Jena7 rules)

```
(?person s:isLocatedAt ?location1) .
(?location2 s:isNear ?location1) → (?person s:isNear ?location2)
```

Next a one line query will serve the purpose:

```sparql
select ?location where (?person s:isNear ?location)
```

In the above case, the query is just for data retrieval, no logic is embodied within it. This approach to get results by querying on inferred knowledge is referred in the paper as rule-based approach.

It is to be noted that the prime purpose of a stream reasoner is to reason as quickly as possible. After initial loading of background knowledge in the Memory Area, the main time consumption is divided among the following steps: a) adding new knowledge b) deleting obsolete knowledge c) firing rules to produce entailment d) querying the combined knowledge. Both addition and deletion of knowledge triggers rule firing. Deletion is an expensive operation because along with the obsolete facts, the obsolete entailed knowledge needs to be deleted. This incremental reasoning is best supported by the Rete algorithm introduced by Forgy [4]. Rete sacrifices memory for speed and hence is suitable in a stream reasoning scenario. Rete builds a network of nodes that represents patterns of conditions in the rules and each of the nodes keep a memory of facts satisfying the pattern. If the maximum reasoning logic is embedded in Rete rules, the queries will be very fast. However if complex logic needs to be embedded in query, SPARQL operators like UNION and OPTIONAL will be needed as logical connectives which are very expensive operations as explained in [14]. Another problem in a query based approach is that the results (virtual entailments) are destroyed after query execution. Such entailments may be a requirement for the success of some other query as we will see in Section 6.2. If the logic of the application is complex, devising queries becomes very difficult. Devising complex rules also becomes difficult from a software engineering perspective. Hence, atomic concept-based rules are used, where a concept is broken down into smaller concepts called atomic concepts that cannot be subdivided. Rules are written based on each such concept. The whole logic of the application is then represented by a set of atomic rules. These rules can be reused in other applications that share the same domain. Also new rules can be added easily without affecting existing rules. Section 6.2 illustrates this with examples. The terms atomic concept-based rules and atomic rules is used interchangeably. The main memory consumed by the rule-based approach is for maintaining the entailments and the Rete network. Queries on the other hand consume memory in its execution state. Once a query finishes, the memory is set free. However, based on the SPARQL semantics in a query the runtime space requirements may be high. The main differences between the rule-based and query-based approach is tabulated in Table 1.

6. IMPLEMENTATION
A prototype has been developed that tests the performance of the stream reasoning system as well as the effectiveness of the proposed smart public alert system. As discussed earlier, the solution is suitable for deployment in smart cities. So a smart city needs to be targeted first. After considerable investigation, the popular city of Washington DC, capital of USA was selected because many real-time feeds are available and it has been accredited by NRDC as a Smart City for.

7http://jena.apache.org/
Transportation. Some other cities where the solution can be readily deployed include San Francisco, Chicago, New York, Singapore and London.

### 6.1 Knowledge Gathering

The main technical challenges in the knowledge gathering process are as follows:

a) gathering accurate data (both city and user profile data)
b) conversion of data into knowledge (ex. unstructured RSS or Twitter posts to RDF)
c) maintaining a service to receive real-time data from alerts and user’s context

The background knowledge consist of data about the city and its citizens. The city data includes street names, zip codes, neighborhoods. The Washington DC locational data was generated using OpenStreetMap, restricted to the city boundary. Another good source to gather geo-location data is GeoNames. Apart from geographical data, the data about the city inhabitants also form the background knowledge. There is an option to enter individual data in a web portal, however readily available citizen data was not found. So for experiments, this data needs to be simulated. The geographic and citizen data is converted to RDF and then reasoning is applied to extend the knowledge using some of the relevant rules mentioned in section 6.2. The main context of the citizen is location which is tracked by their GPS text of the citizen is location which is tracked by their GPS

### 6.2 Methodology

The main target is to get a list of citizens and the corresponding events that have affected them. If the entailments contain such a pattern then the following query will suffice:

```sql
SELECT ?person ?event
WHERE {
  ?person p:isAffectedBy ?event
}
```

Desired example output to the above query will be of form: ‘John Wayne’, ‘Traffic Disruption at X street’

Some of the atomic concepts that will eventually entail knowledge patterns to support the above query are:

**Concept:** User’s Preference for Alerts based on Event Type

**Rule 1:** (?user p:hasPref ?eventType) → (?location1 s:hasZip ?zip)

**Rule 2:** (?location1 s:isNear ?location2) → (?user p:hasPref ?eventTypeSub)

**Example:** If a user has preference for Traffic Disruption event, he has preference for all of its subclasses like shooting, accident, road blockage, traffic jam, etc.

**Concept:** Neasrness of a place

**Rule 1:** (?location1 s:hasZip ?zip) → (?location2 s:isNear ?location1)

**Rule 2:** (?location1 s:hasZip ?zip) → (?location1 s:isNear ?location2)

**Example:** If two locations share the same zip (or postal code) or neighborhood they are near to each other.

**Concept:** A place is Near or On a place

**Rule 1:** (?location1 s:isNear ?location2) → (?location1 s:isNearOrOn ?location2)

**Rule 2:** (?location1 s:isOn ?location2) → (?location1 s:isNearOrOn ?location2)

**Example:** (X is located on Y) and (X is located near Y) is combined to form a single property. This is useful if the

---

15. [http://twitter.com/#!/wmata](http://twitter.com/#!/wmata)
UNION { ?event s:atLocation ?location2.
  { ?event s:atLocation ?location1. }
  ?person s:willBeAtFutureLocation ?location1.
  select ?person ?event where {
    ?event s:hasLocation ?location1.
    {?user p:hasLocation ?location1} .
    {?person p:hasPref ?event} .
    {?event e:ofType ?eventType} →
    {?event e:affects ?user} UNION
    {?event e:indirectlyAffects ?user} }

Concept: An event affecting a user based on profile

Rule: (?event p:hasDisease ?disease) .
  (?event p:hasAdverseEffectOn ?disease)
  → (?event e:affects ?user)

Example: If a user has heat related disease, a hyperthermia alert will affect the user.

Concept: An event affecting a user’s relatives

Rule: (?user1 p:hasRelation ?user2) .
  (?event e:affects ?user2)
  → (?event e:affects ?user1)

Example: If a child is affected by an event so is his father. The condition ‘User has interest in certain event types’ can also be added to the above rule in the way discussed earlier. However there is a problem in the last rule, because the rule is transitive. The rule entailment should be restricted to one level. Otherwise there is high chance that generated entailments will falsely say that every person will get affected by any event. This assumption is based on the theory of six degrees of separation [6] where it has been shown that the average distance between two nodes (persons) in the social graph (graph connecting persons based on acquaintance) is usually less than six. So the last rule is redefined with a different property ‘indirectlyAffects’:

Concept: An event indirectly affecting a user

Rule: (?user1 p:hasRelation ?user2) .
  (?event e:affects ?user2)
  → (?event e:indirectlyAffects ?user1)

The initial query has to undergo a change to accommodate the above rule: select ?event ?person where {
  {?event e:affects ?person} UNION
  {?event e:indirectlyAffects ?person} }

Another way is break this into two separate queries. As the final goal is to search for graph patterns <e:indirectlyAffects> and <e:affects>, the search space can be restricted to entailments only.

If rule-based approach was not used, then a query that matches events based on location will look like:

select ?person ?event where {
  ?person s:willBeAtFutureLocation ?location1.
  {?event s:atLocation ?location1. } UNION
  {?event s:atLocation ?location2.}

7. EVALUATION

The solution has been developed on the Java platform and the experiments were run on an Intel Core2 Duo Machine with processor speed of 3 GHz and memory size of 2 GB. The background knowledge consisted of extended information about 1835 streets of Washington DC and simulated profile data of citizens. Context and alert information was simulated similar to actual alerts and context. Comparison of atomic rule based approach with query based approach was carried out. To verify the efficiency of the approaches, logic was embedded in a query as shown below and its corresponding atomic rules and query were then created.

a) Query based Approach:

http://linkedevents.org/ontology/
http://www.foaf-project.org/
http://vocab.org/places/schema.html
http://revelytix.com/content/when-write-rules-and-when-focus-ontology-development
http://developers.facebook.com/
https://developers.google.com+/api/
http://maps.google.com/help/maps/directions/
http://www.microsofttranslator.com/dev/
In the experiment regarding comparison of query based and atomic rule based approaches, the average time for processing events were 220 ms and 20 ms respectively. For the query based approach the time spent is for updating the Memory Area and querying. For the rule-based approach, time is spent on querying, adding and deleting triples and rule firing. Rules are fired when events enter or exit the system. It was seen that updating an event into Memory Area took more time for rule-based approach. The reason for that is for every update, rules were triggered where as for query based approach, rules are not used in the first place. As the UNION operation is an expensive SPARQL operation and as fast Rete was used for atomic-rule based approach, the performance was an order of magnitude better for the latter approach. It was also observed that as the time spent on querying in the query-based approach was high, many events were kept waiting in the queue in the Event Manager. In the rule based approach the queue was empty most of the time as events were getting added to the Memory Area very fast, as very little time was taken by the small query to execute. The system was tested with various loads of concurrent dynamic data and the results were satisfactory. The average time for an event to be sensed and sending notification to end-users through the system was within a few seconds and hence acceptable in a public alert scenario.

8. CONCLUSION

In this paper we have described a smart public alert system that sends only relevant alerts to citizens of a smart city. This is achieved by maintaining and managing knowledge about the city, its events and its citizens’ social profile, preferences and contextual information. How stream reasoning fits well into the problem has also been discussed. A solution architecture based on stream reasoning to address the problem has also been described. Comparison between query-based approach and atomic-rule based approach for stream reasoning is conducted and the results show atomic rule based approach as a better choice. A prototype of the proposed system has been developed that is ready for pilot studies in smart cities.

9. FUTURE WORK

After a citizen gets an alert that he or she will be affected by some event, what action should the citizen take in response? This issue will be addressed by real time planning in the future work. Another important scope of work is to apply machine learning techniques to model user behaviour like what places the citizen goes usually and based on that forward alerts. Optimizing the performance of the stream reasoning system is yet another important scope of work. Applying machine learning and doing natural language processing on unstructured data about events and context is also another area of work. Pilot studies of this intelligent application on other smart cities will be carried out to find its ease of extensibility.

10. REFERENCES


