2010 Sixth International Conference on Signal-Image Technology and Internet Based Systems

Event Based Modeling for Context-Reactive Information Systems

Ansem Ben Cheikh
Laboratory of Informatics of Grenoble
Grenoble, France
Email: Ansem.Ben-Cheikh@imag.fr

Agnès Front
Laboratory of Informatics of Grenoble
Grenoble, France
Email: Agnes.Front@imag.fr

Stéphane Coulondre
INSA Lyon, LIRIS
CNRS France
Email: Stephane.Coulondre@insa-lyon.fr

Jean-Pierre Giraudin
Laboratory of Informatics of Grenoble
Grenoble, France
Email: Jean-Pierre.Giraudin@imag.fr

***Abstract*—Ubiquitous Information Systems (ISs) should support mobility of users and reason efficiently on their context to provide relevant and real-time information. ISs should be particularly adaptive and context-aware by managing and responding properly to occurring events. A need of integrating an adaptive and distributed event based system raises to lead to a diversity of proposed approaches and architectures. Unfortunately, the high level link between context and events is not explicitly considered in these systems. In this paper, we propose a metamodel coupling context and event metadata. The metamodel highlights some requirements considered in the definition of an event processing architecture to maximize communication security between distributed components. Context-aware event processing is a natural result of such architecture enabling the integration of context reasoning in event based systems. The proposal is used to implement an assistant prototype for passengers in the public transportation domain.**

***Keywords*-context awareness, event processing, personalization, metamodel, distributed architecture.**

I. Introduction

Ubiquitous information systems (ISs) are context-aware systems supporting mobile applications. Mobility of users induces new requirements presenting new challenges that must be considered and handled by the IS. Reporting events occurring in the users operating environment is an essential step for mobile applications to be aware of their context. Different application fields are interested in managing events like financial market analysis, trading, security, customer care services. These fields need to react efficiently and in real-time to events especially to handle exceptional situations or to indicate opportunities and risks. A passenger assistant application in the transportation field is an example of customer care services. To enhance the use of public means of transport, it is essential to provide the user with accurate and real-time information. The user should rely on the IS especially in perturbation situations. Here events play a crucial role. The more the system can respond rapidly and efficiently to occurring events in the environment, the more the provided information will be useful and fresh. Context information is collected from different sources (GPS, sensors, Bluetooth, etc). They may be in other cases provided by roles interacting with the IS by notifying events or updating internal data like decision rules, policies or databases. Knowledge on context could be used for two different purposes. First, it is used for personalization issues by providing services adapted with the context and the profile of the user. Secondly, it is used for pro-activity issues by being aware of changes occurring in the context and reacting appropriately to events.

This paper focuses on context-reactive IS, where an appropriate event processing architecture is needed to guaranty application flexibility and responsiveness. The goal of this paper is to study the use of events for both purposes cited above while guarantying security and property of personal data. As a consequence our approach is built upon the following hypotheses:

- saving locally user context and profile in their mobile device or proper processing units in order to enhance personal data security,
- filtering information according to context and profile data in order to personalize information delivery,
- sensibility to ambient events from different sources in order to collect context data,
- adopting an event oriented approach to guaranty a near real-time or a short time response. Realtime is an approximate performance measure that depends of several considerations in application design.

These hypotheses lead to a need of combining event processing and context reasoning. We notice a lack of event models supporting contextual data and their use in event processing methods. This paper tries to raise this challenge by proposing a metamodel coupling event and context metadata. The structure of the metamodel is used in the design of the system architecture. The ubiquitous nature of the system imposes the use of distributed event processing units. But the need of global reasoning on context implies the need of enhanced communication between processing units. Hereafter, the concept "Resource" refers to each entity interacting with the IS by using or providing context information. Each resource has an event processing unit responsible for detecting and delivering events to it. The architecture highlights resource monitoring to detect events and context data and then, to deliver events.

The remainder of this paper is organized as follows. Section 2 represents a state of the art on some related works proposing event and context models. Other related works on architectures for context-aware and event based systems are commented as well. The proposed metamodel for coupling events and context is proposed in section 3. Section 4 deals with proposed high level and internal architecture for event processing systems. Section 5 presents an implementation tool for the proposed architecture while section 6 presents concluding remarks and future work.

II. Related work

Several context models and several adaptive and context-aware applications have been developed and implemented. however, there exist little works interested in context metamodels [5][16][21]. It is worth mentioning that these works associate profiles for human users and decompose the context into multiple dimensions. This needs a preliminary knowledge of the application domain to specify the appropriate profile dimensions set. Our approach has the advantage of defining a generic context metamodel to be instantiated with the considered domain.

From another perspective our work has interest in structural and abstract views of event metadata independently from executive platforms. Several works propose structural event metamodels [2][11][12][14][20]. The works provide interesting architectural event views [2] and event concepts [11][14][20]. But the main shortcoming of these works is the lack of context concepts in their metamodels.

On the coupling of event models and context models, few works were done. We cite the papers [18] and [1]. In [18] a framework used to adapt with mobile challenges (bandwidth, connectivity, memory) is proposed. The framework contains a mobile event engine for event detection and event delivery and a rule system to guaranty adaptability by considering user profile. The main lack in this work is a higher view and models showing links between context and event processing. In [1] a set of interesting coupling models is proposed to type context and define context attributes. Context acquisition is done by sensors, while in our approach context is constructed with occuring events from any source. There is a second constraint on notification as it is only possible after subscription to particular context types. Moreover the concept of event is not visible in models (use of sensors, notifiers) and they are not directly linked to application entities. Finally, no link with event processing is evoked.

Existent architectures in the literature vary according to the intent and the objective of target systems and domains. It is essential to study these different systems and focus on context oriented approaches to identify the shortcomings of these systems in handling our hypotheses requirements cited in the introduction. In general the architecture of a context aware system is defined according to global context reasoning strategies. Therefore, context-aware systems are often built upon a centralized architecture consisting of a context manager that collect context data from distributed systems to process them in a single service containing a reasoner component [13][17][19][5].

In the Event Driven Architecture (EDA), two trends exist today to define the structure of an event processing system: centralized event processing and distributed event processing. Centralized event processing systems consist of a single processing unit where all enterprise events are processed. Distributed event sources communicate with the centralized unit to provide events. The unit processes events and notifies consumers. Works like [20], [18] and [6] propose centralized system architectures. This trend is useful to guaranty the completeness of event processing as it is possible to consider all occurring events before reacting. It optimizes as well the response time by treating events with priority orders. On the other hand, centralized event processing is hard to implement as all event sources, internal or external, and all event consumers should be related to the system. It is also complex to add new functionalities to the event processing system as all sets of rules could be reconsidered and modified.

The use of distributed event processing systems is more adapted with ubiquity requirements and mobile applications. This architecture style is a good solution, but it needs techniques to guaranty coherence between processing components. Moreover, the link between event processing and context reasoning is not handled by these distributed solutions. Examples of distributed approaches are shown in the specification of distributed Event Processing Networks like in [9] and [7].

This paper presents a distributed architecture taking profit from different architectural styles cited above and linking event processing to context reasoning which is illustrated with a generic metamodel coupling event and context concepts. Designing conceptual models linking these two research areas and supporting an appropriate architecture for context-reactive ISs was not handled before which highlights the originality of our approach.

III. A metamodel coupling events and context

Events are used in ISs to react in timely manner to changes in the context of the system. It is essential to use event models combined with context models to depict their mutual influence. As a matter of fact, we propose a generic metamodel (see fig. 1) that exhibits links between events and context. The metamodel consists of 3 views that correspond to: the context metamodel (view A), the event metamodel

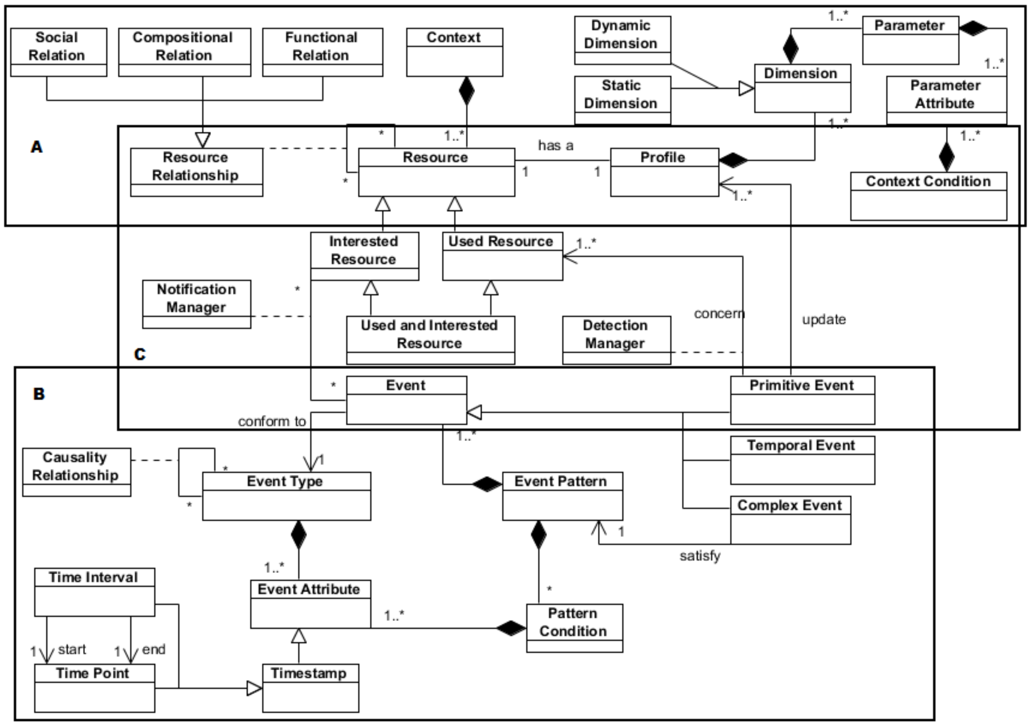


Figure 1. The coupling metamodel between event and context views.

(view B) and the links between context and event metadata (view C).

A. *Context View (view A)*

Concepts like environment and context are commonly used, although their meanings are sometimes interchanged. Thus it is essential to give appropriate definitions for these concepts widely used in this paper. We define an environment of an application, a system or an entity as the set of surrounding elements in a specific situation. Several definitions of context are given in the literature, but we adopt the definition given in [4]: "a context is 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 applications themselves".
In this proposal a context metamodel describing the different resources according to a single template is needed. Therefore, we propose the context metamodel depicted in the view A of fig. 1. To facilitate context reasoning the metamodel assembles environmental entities that could be sensed and observed and that could interact with the system under the concept "resource". The context is the set of resources and a profile is associated to each resource. The profile consists of dynamic and static metadata of the resource. In the transportation field, resources like vehicles, stations, drivers and users have their own profile. For instance the current itinerary of a passenger is a dynamic profile dimension while usual itineraries (home-work, home-school, home-sportclub) are static profile dimensions. Only dynamic profile dimensions could be supervised to observe state changes (localization of a user or a bus, number of passengers on a bus or at a station). Static data are rarely modified and this could be done only manually by the administrator of the application (the user). For example, the set of bus lines serving a station is a static profile dimension of the station that changes only after a global or a partial modification of the transport network.

As mentioned above, the profile concept is generally related to a human user. However, in our proposal a profile is associated to each resource communicating with the system. The idea behind this is to facilitate communication between resources and keep traces of events through updates performed on the profile. Then context could be built when needed by assembling resource profiles.

The metamodel should also provide resource relationships used to derive the influence of a resource event on related resources. In [21] relations between entities of the context could be: functional, compositional or social. The metamodel maintains this classification of resource relationships. When two resources are related, the event concerning the one may interest the other. For instance, the event "accident" concerning a precise road may interest the different transport vehicles using this road in their journeys.

B. Structural view of events (view B)

The event view in the metamodel, depicted by view B of fig. 1, assembles the adopted concepts from Complex Event Processing (CEP) research area. According to [10] an event has three aspects: form, significance and relativity. The form of the event is an object having particular attributes or data components. The form of an event is also called event type or event schema and attributes could be named properties. Significance of an event points out its meaning. An event may report a state transition, an activity or just an interesting occurrence in time. Relativity in [10] means relationships between events that could be time, causality and aggregation. View B of fig. 1 depicts causality and aggregation (by the use of complex events) relationships between events. Each event has a timestamp that may refer to occurrence or detection time, according to system and application nature. Timestamps could be considered as attributes, but here they are directly related to events as they are widely used to order events and to deduce temporal relationships between them.

Events are classified into primitive and complex events. In [3] "primitive events are specific to a domain and are predefined in the system. They are assumed to be detected by the underlying system along with the time of occurrence". A complex event is an aggregation (composition) of other events (primitive or complex) [10][3]. The concepts simple and composite events are sometimes used to refer respectively to primitive and complex events. Temporal events constitute another type of events and they may be absolute or relative. Absolute temporal events are specific points in time while relative temporal events are defined according to another event (relative temporal event = event + time interval).

An event pattern is a "template that matches certain sets of events. It describes precisely not only the events, but also their causal dependencies, timing, data parameters and context" [10]. An event pattern contains a set of events linked by logical and temporal operators to which a pattern condition is added. Pattern conditions define constraints on event attributes. For example, an event pattern could report disturbed traffic conditions if an event indicating slippy roads within city center is associated with a temporal event indicating rush hours.

C. Conceptual coupling between events and context

The coupling between event and context metadata is depicted in view C of fig. 1. The context is a set of resources that have two possible usage modes of events: event producing and event consuming. Then context is related to event through two possible links: event detection and event notification. The coupling view distinguishes between different types of resources according to their ability to produce or to use events:

- **Interested resources**. They are resources interested in certain events produced by the system. The system, when delivering events to these resources, should take into account their context in order to deliver only useful information. Stations, public transport vehicles and users are examples of interested resources.
- **Used resources**. They are event providers. The system is generally interested in these entities as any change in their state may impact the system functioning. Transport vehicles, weather services and traffic services are used resources. An event concerning a resource induces a state change that should be considered by updating the profile of the corresponding resource.
- **Interested and used resources**. This type is the intersection of both resource types cited above. A user assisted by the IS during his itinerary should be located and according to his situation notification events are sent to him. Other resources like public transport vehicles and drivers are interested and used resources.

Detection component deploys techniques and strategies to detect events by supervising the concerned resources. Detected events may be reported by the resource itself or using sensing and observation techniques. Similarly, notification component deploys techniques to notify interested resources. When notifying interested resources several context data should be considered like profile parameters and resource relationships. Detection and notification components are parts of the system architecture described in the next section.

IV. System architecture

For pervasive services in general and for the passenger assistant application in particular, a number of design choices supporting autonomous event processing capabilities and enhanced reactivity to context are needed (see section I). This section explains the high level architecture of the proposed event processing network consisting of processing units. Then we explain the internal architecture dealing with units structure.

A. Event Processing Network Architecture

In context-aware systems the event processing architecture should provide distributed components that perform three functionalities:

- collecting context information from context resources collocated with them.
- allowing an initial event processing phase. This phase is used to generate only useful events to be sent to interested entities to minimize unnecessary event flows.
- delivering information to collocated context resources if needed.

Fig. 2 depicts an example of an event processing network used to define the event processing system in the public transportation field. Event processing units communicate with resources and are responsible for detecting events from the resources and delivering events to them if needed. Some units are integrated in the IS, while others are implemented

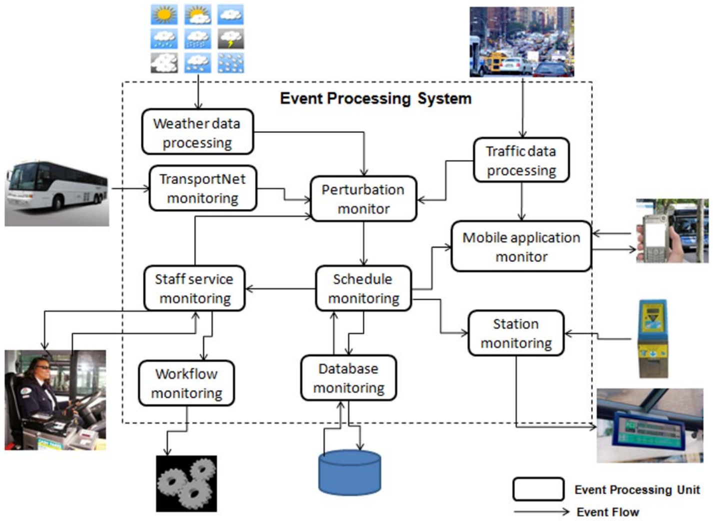


Figure 2. An example of an event processing network in the transportation domain.

in distributed applications like user mobile application, station infrastructure and vehicle embedded systems. Units like weather data processing, traffic data processing and transportNet monitoring are included in the system and communicate with external event sources (RFID sensors used to detect vehicle position on track, traffic and weather services). Monitors are associated also to internal components of the IS like workflows and databases to report interesting events to trigger business processes or update data.

An Event Processing Network (EPN) was first mentioned by Luckham as defined in [15], consists of four components: event producer, event consumer, event processing agents and event channels (connection components). An EPN describes how events flow from producers to consumers and how they are processed. As we are dealing with ubiquitous systems, our EPN contains large distributed event processing components called Event Processing Units (EPUs). Event processing agents and event channels are lower level components that are not described for clarity reasons.

By generalizing the architectural choices shown in fig.2, this paper proposes the use of an integrated system relating distributed event processing units. The event processing system manages communication between units. Units communicate by exchanging event flows. Each processing unit is considered as a service and the integrated system is responsible of managing access rights between them. Fig. 3 depicts a class diagram representing the different components of the architecture and their links. Two complementary and essential communication techniques are deployed:

- Subscription. External resources can express their interest in some events by subscribing (subscription rules) for particular types of events. They may impose some conditions on event attributes like “inform me only of incident events occurring on a specific geographical area”. Then, it is possible to subscribe to different event types (primitive, complex) or even subscribe to event patterns. In mass delivery, some types of events can be sent massively to specific user categories. Users will receive the same information and can filter by themselves the information with their event processing units.

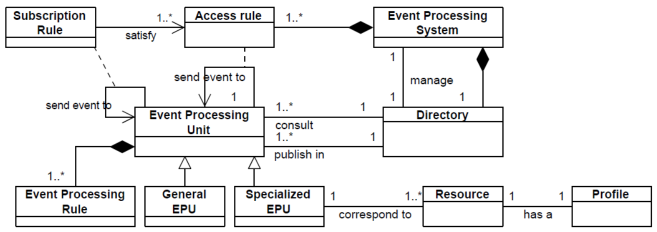


Figure 3. The architecture model.

- Directory discovery. The directory contains information on functionalities of each unit and their products. It enables event customers to locate interesting events and discover the units producing them. When a new event is generated by an EPU, the latter should publish it in the directory to be consulted by other EPU. Only the event type should be published, and event details could be accessed if the interested EPU has an authorization. The use of a directory in this context is similar to the use of the UDDI directory in the Service Oriented Architecture to publish and describe web services. Using a directory enhances flexibility of the system by adding and removing easily EPUs while being visible for other resources.

Access rules define access rights and denied accesses of EPUs to different event types. We define general access rules depicting access rights of a certain EPU to another. In this case the farmer EPU has authorization to all events produced by the latter one. We deploy an RBAC (Role Based Access Control) solution, known for its flexibility as authorization are assigned to units according to their role names. A hierarchy used to organize role names facilitates the definition of access rules and inheritance access rights between roles.

An EPU could be general or specialized. A general EPU is used to process complex events using context data. It uses CEP techniques and communicates with other EPU to collect events. This type of EPU is used to centralize some processing work by creating a processing unit independent from other internal or external system resources. A specialized EPU is directly related to a resource or a category of resources. It is responsible for detecting events when supervising the associated resources and delivering interesting events to them according to their context.

B. *Event Processing architecture*

In the previous section, we proposed a general view of the system architecture. In this section we detail the internal processing architecture of EPUs. As the system manipulates two different types of EPUs performing common and different functionalities, there are two lightly different architectures, one for each type.

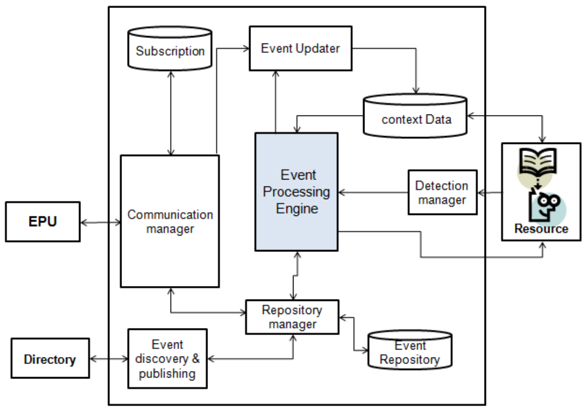


Figure 4. Architecture of a specialized EPU.

As shown in fig. 4, a specialized EPU is externally related to other EPUs. It stands for a set of system resources. Thus received events could be obtained from two different ways: detecting resources events and exchanging event flows with other EPUs. The central component of an EPU is the event engine, who runs continuously event patterns to detect firing situations. An event pattern may trigger several kinds of rules expressed in the Event Condition Action form:

- event processing rules : producing intermediary events to be used in a processing chain or to be published in the directory as products.
- update rules : producing update events used for updating the contexte database by the event updater component
- notification rules : used to reason on context to notify the associated resource by producing notification rules. Queries expressed by users are processed as notification rules.

When an event is needed, a request is sent to the repository manager to look for the desired event in the event repository (containing historical and produced events saved according to predefined policies). If the event is not present in the event repository the request is delegated to a component called "event discovery and publishing" that could consult the directory and query it to detect the desired event. To obtain a version of the event, a request is sent to the integrated system that verifies access rules and creates an event channel between the two extremities of the event flow. When the event is detected through the communication manager, it is sent to the event repository who routes it to the engine and decide whether or not it should be saved in the event repository. If the EPU is subscribed to an event it will receive it automatically through the communication manager. The latter manages also subscriptions of other EPUs to some products of the EPU. It identifies from the subscription repository EPUs subscribed to a produced event in order to deliver events to them through the event channels built between the EPU and its subscribers. Events produced by processing units may be sent to internal computing layers like workflow engine, active databases or network monitor.

In this event driven solution, different activities and changes are considered as events. The detection manager has to collect events concerning associated resource from heterogeneous sources (sensors, GPS, interaction with users) and routes them to the processing engine. When some events satisfy an event pattern reporting a resource state change, an update rule is triggered. Then, an update event is created and is sent to the "context updater" component, responsible for updating the context database.

In specialized EPUs, the event engine uses profile and context data to filter events and save only interesting ones. Notification rules express situations where the resource should be alerted. When an event pattern triggers a notification rule the engine uses context data to decide whether to alert the resource according to its situation or its profile. For example a schedule delay of a train should not be sent to stations not disserved by the considered line.

For a general EPU, the communication and the event processing components have the same architecture as a specialized EPU. The only difference consists in communicating with resources as general EPUs are not considered by collecting events from resources and delivering events to them. However the context database should remain as it contains a vision of the execution environment.

V. System implementation

We built a passenger assistant application communicating with its environment by sending and receiving events. The user is assisted while using an itinerary or only to consult transport information. The application is a prototype where dynamic data (events) are entered manually for simulation goals. The prototype could be exported to be used on a mobile device.

A. *Application Design*

As there exist a wide variety of event processing solutions in the literature, we decided to adopt an event processing engine compatible with our vision of event processing, that is described in more detail in section IV-B. We adopted the Esper event engine [8] that is designed for Complex Event Processing (CEP) and Event Stream Processing (ESP). Esper is extended with event management and context reasoning component as depicted by figure 4. In Esper queries are specified in a SQL-like language called Event Processing Language (EPL). While EPL has the usual features of a data stream language, it has been extended with pattern-matching constructs, inspired by composition operator- based languages. The Esper engine provides also an interesting feature, as it is possible to query external data and to combine results with usual queries on events. This feature facilitates context reasoning by accessing the context database. Suppose that a user is interested in an event "schedule delay" only if he has in his profile (especially in his transport habits) an itinerary using the corresponding bus line in the same station and at the same time. This example is performed in EPL as follows.

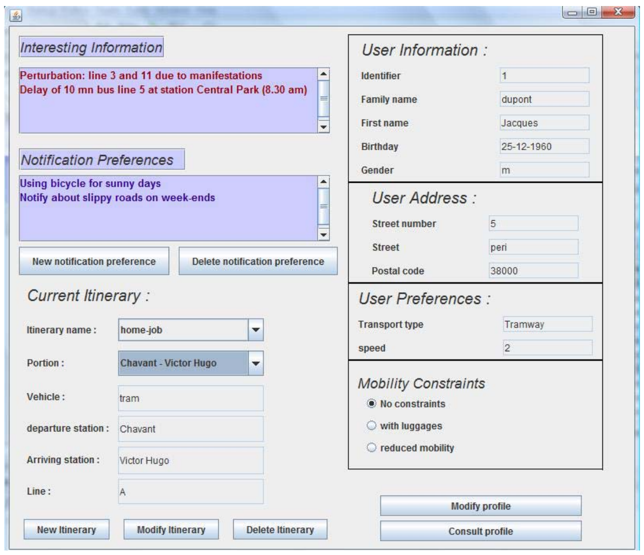
select profileId
from delayschedular as e
sql : ContextDB ['**select** v, profileID
from itinerary.vehicle **as** v, profile.ID **as** profileID']
where e.vehicle = v

The application contains an Oracle database saving user profile (static data, transport preferences, habitual itineraries, adresses for home, job and university, job schedule, mobility constraints,etc) as well as dynamic profile (GPS localization and transportNet localization : in a bus or at a station). The application communicates by Bluetooth with surrounding devices to receive events on traffic, weather and perturbations of the transport network. We saved a set of rules (notification, processing and update rules) combining different event types and conditions on their attributes as well as context conditions and the events to be generated if the rule matches. The application contains also a database for theoretical transport data like vehicules, habitual journeys and schedules to allow a personalized itinerary calculations.

In Esper event types are saved as Java objects, XML files or Map types. We used the first possibility by developing each event type as java class. Thus, the designer could specify event types and their attributes manipulated by the event engine and an associated class is generated and added to Esper configuration. The designer has the possibility to define rules used to process events, notify the user and update his context. These rules are called statements. As Esper has not the possibility to modify an external database, we developed the context updater component who receives update events and analyzes them to update the concerned database using an OJDBC module.

B. *User view*

From the passenger perspective, in a first connection the user should fill in profile information by specifying personal information as well as habitual transport practices. These data with other dynamic profile dimensions (sensed localization in the transport network or by GPS) compose the user context. Fig. 5 shows the application's home window developed with Eclipse. The user has the possibility to define a set of rules reporting his information preferences. For example the user could specify if the assistant should recommand using bicycle to get to work by sunny time. These rules are considered as notification rules saved in Esper and using context and profile data. They are executed simultanuously with other notification rules specified by the designer.



Interesting Information
Perturbation: line 3 and 11 due to manifestations
Delay of 10 mn bus line 5 at station Central Park (8.30 am)

Notification Preferences
Using bicycle for sunny days
Notify about slippy roads on week-ends

Current Itinerary :

Itinerary name :	home-job
Portion :	Chavant - Victor Hugo
Vehicle :	tram
departure station :	Chavant
Arriving station :	Victor Hugo
Line :	A

User Information :

Identifier	1
Family name	dupont
First name	Jacques
Birthday	25-12-1960
Gender	m

User Address :

Street number	5
Street	peri
Postal code	38000

User Preferences :

Transport type	Tramway
speed	2

Mobility Constraints
[✓ No constraints] with luggages reduced mobility

Figure 5. An interface from the passenger assistant prototype

The prototype is simulated by sending manually events concerning perturbations, user localization, traffic and weather data. The update, processing and notification rules allow producing new events, updating the context database and providing the passenger with personalized messages. For example only perturbation concerning transport vehicules used in habitual itineraries are displayed on the interface.

In a future work, the prototype will be implemented and tested on a mobile device.

VI. Conclusion

Event distributed systems and ubiquitous systems share common properties like the need of managing external events and informing properly users. This challenge was raised in this paper by considering context in event processing systems. The main contribution of this paper deals with using contextual information for both event processing and delivery. This goal was supported with a distributed architecture built according to a conceptual metamodel consisting of three views. In our approach we consider all entities related to the IS (using or providing context information and events) as system resources. The system architecture consists of distributed event processing units. The proposed event driven architecture is enriched with components like a directory and a set of access rules to enhance communication between distributed units. We propose also a set of components to enrich event processing units by deploying context reasoning. This architecture was applied in designing a passenger assistant application used in the public transportation domain. As this approach handles requirements of ubiquitous ISs, our goal, in further works, is to study the implementation of the architecture and the supporting metamodel in a global engineering process. Moreover context management will be improved by defining new techniques for context composing and sharing using annotated and enriched events (with context data).

Acknowledgment

This work is part of the DéSIT project (http://projet-desit.imag.fr) which is supported by the TTS cluster of Région Rhône Alpes (France).

References

[1] R. Belotti, C. Decurtins, M. Grossniklaus, M.C. Norrie, A. Palinginis, "Modelling context for information environments", Ubiquitous Mobile Information and Collaboration Systems, pp. 43-56, Springer-Verlag Berlin, 2004.

[2] R. Blanco, J. Wang, P. Alencar, "A metamodel for Distributed Event based Systems", Distributed Event Based Systems, Rome Italy, ACM, 2008.

[3] S. Chakravarthy, Q. Jiang, "NFMi: an inter-domain network fault management system", Stream Data Processing: A Quality of Service Perspective, Springer, US, pp. 167-186, 2009.

[4] A. Dey, "Understanding and using context", Personal Ubiquitous Computing, Springer, London, 2001.

[5] P. Dockhorn Costa, L. Ferreira Pires, M.J. Van Sinderen, "Designing a configurable service platform for mobile context-aware applications", Pervasive Computing and Communication, vol. 1, n1, pp. 13-23, 2005.

[6] J. Dunkel, A. Fernandez, R. Ortiz, S. Ossowski, "Event Driven Architecture for decision Support in Traffic Management Systems", International Conference on Intelligent Transportation Systems, IEEE, 2008.

[7] G. Eisenhauer, H. Abbasi, M. Wolf, K. Schwan, "Event Based Systems: Opportunities and challenges at exascale", Distributed Event Based Systems, Nashville USA, ACM, 2009.

[8] Esper, Event Stream Intelligence. Available: http://esper.codehaus.org/

[9] G.T. Lakshmanan, Y.G. Rabinovich, O. Etzion, "A stratified approach for supporting high throughput event processing applications", Distributed Event Based Systems, Nashville USA, ACM, 2009.

[10] D. Luckham, The power of events: an introduction to complex event processing in enterprise systems. Addison Wesley, 2002.

[11] A. Nagargadde, S. Varadarajan, K. Ramamritham, "Semantic characterization of real world events", International Conference on Database Systems for Advanced Applications, Springer-Verlag, Berlin Heidlberg, 2005.

[12] A. Paschke, P. Vincent, "A reference architecture for event processing", Distributed Event Based Systems, Nashville USA, ACM, 2009.

[13] T. Patkos, A. Bikakis, G. Antoniou, M. Papadopouli, D. Plexousakis, "A Semantics-Based Framework for Context-Aware Services: Lessons Learned and Challenges", International Conference on Ubiquitous Intelligence and Computing, Springer-Verlag, Berlin Heidlberg, 2007.

[14] S. Rozsnyai, J. Schiefer, A. Schatten, "Concepts and models for typing events for Event Based Systems", Distributed Event Based Systems, Toronto Canada, ACM, 2007.

[15] G. Sharon, O. Etzion., "Event-processing network: model and implementation", IBM System Journal, **47**, pp. 321-334, 2008.

[16] C. Simons, G. Wirtz, "Modeling context in mobile distributed systems with the UML", Journal of Visual languages and Computing, vol. 18, pp. 420-439, Elsevier, 2007.

[17] M.J. Van Sinderen, A.T. Van Halteren, M. Wegdam, H.B. Meeuwissen, E.H. Eertink, "Supporting context-aware mobile applications: an infrastructure approach", IEEE Communication Magazine, vol. 44, n9, pp. 94-104, 2006.

[18] S. Wu, C. Chang, S. Ho, H. Chao, "Rule based Intelligent adaptation in mobile information systems", Expert Systems with Applications, vol. 34, pp. 1078-1092, Elsevier, 2008.

[19] K. Xu, M. Zhu, D. Zhang, T. Gu, "Context aware content filtering and presentation for pervasive and mobile information systems", Ambi-Sys, Quebec Canada, 2008.

[20] C. Zang, Y. Fan, R. Liu, "Architecture, implementation and application of complex event processing in enterprise information systems based on RFID", Information System Frontier, vol. 10, pp. 543-553, Springer, 2008.

[21] A. Zimmermann, A. Lorenz, R. Oppermann, "An operatioanl definition of context", Modeling and using context, pp. 558-571, Springer Berlin Heidelberg, 2007.