Context Management in Event Marketplaces

Yiannis Verginadis(✉), Ioannis Patiniotakis, Nikos Papageorgiou, Dimitris Apostolou, and Gregoris Mentzas

Institute of Communications and Computer Systems,
National Technical University of Athens, Athens, Greece
{jverg, ipatini, npapag, dapost, gmentzas}@mail.ntua.gr

Abstract. This paper refers to methods and tools for enabling context detection and management based on events. We propose a context model that builds on top of previous efforts and we give details about the mechanisms developed for context detection in event marketplaces. In addition, we show how simple or complex events can be used in combination with external services in order to derive higher level context with the use of Situation-Action-Networks (SANs). Specifically, we present two different approaches, one for detecting low level context and another one for deriving higher-level contextual information using SANs. We present an illustrative scenario for demonstrating the process of specialization of our generic context model and its instantiation based on real-time events.

Keywords: Context · Event marketplace · Detecting context · Deriving context

1 Introduction

Context is "any information that can be used to characterize the situation of an entity, i.e., a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [1]. Context detection is considered important in the so-called event marketplaces [2] (see e.g., https://Xively.com, a platform offering a service based architecture, a range of graphing and visualization tools, event detection via triggers, along with cost-effective data storage) for enhancing the user's experience when interacting with the event marketplace.

Events from event marketplaces are an important source of context for service-based applications that consume them because they may convey important information, which is relevant for service execution and adaptation. To achieve the goal of injecting event processing results to context, an event-based context model is needed along with context detection and derivation mechanisms. In previous work [3] Situation-Action-Networks have been proposed as a hierarchical goal-directed modeling approach comprising nodes with specific semantics used to model goal decompositions, enriched with flow control capabilities. SANs provide means to decompose goals into subgoals and capabilities for seeking and achieving the high-level goals, involving situations (i.e. complex event patterns), context conditions and actions.

E. Simperl et al. (Eds.): ESWC 2012 Satellite Events, LNCS 7540, pp. 313–326, 2015.
DOI: 10.1007/978-3-662-46641-4_24

The simplest SAN possible is a two level tree with a parent (root) node and three child nodes, each of them having specific semantics. A parent node models the Goal sought. The leftmost child node describes a situation that must occur, in order to start goal seeking. The middle child node corresponds to context update and requires that a specific contextual condition is true before continuing with the SAN traversal. The rightmost child node specifies the action to be taken in order to fulfil the goal. Rightmost node can also be a sub-goal node with its own three child nodes, or it can even be a construct joining several sub-goals in sequence or in parallel. As the SAN becomes more complex, involving several subgoals (Fig. 1), it deepens and reveals its hierarchical and goal-directed characteristics. In this work, SANs are extended so that they can be used for detecting and deriving context from events.

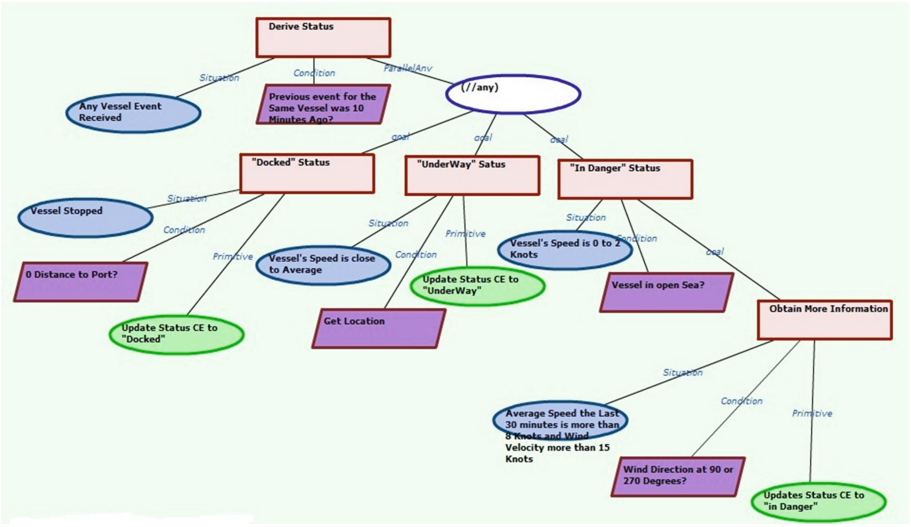

Fig. 1. SAN Illustration based on the marine vessel traffic scenario: Pink nodes denote Goals, Blue nodes denote Situations, Magenta nodes denote Context Condition and Green nodes denote Actions (Color figure online)

This paper continues with a discussion about related work in the domain of event-based context management, while in Sect. 3 it presents a generic context model that is considered appropriate for the needs of event marketplaces. In Sect. 4, we consider two different approaches, one for detecting low level context and another one for deriving higher-level contextual information using SANs. In Sect. 5, we show how the generic context model can be specialized so that it can be instantiated to support an example scenario. We conclude in Sect. 6 with a summary of our event-based context management approach.

2 Related Work

Context-awareness in service-oriented systems refers to the capability of a service or service-based application to be aware of its physical environment or situation and to respond proactively and intelligently based on such awareness; see e.g. [4]. Through

the use of context, a new generation of service-based applications is expected to arise for the benefit of coping with the dynamic nature of the Internet; see e.g. [5, 6]. The multiplicity of applications and the surge of research activities in context-aware service systems are also evident in recent survey research; see e.g. [7–11] point out that in the case of service-based applications context has various different facets as it includes information ranging from the situation in which users exploit a service-based application to the conditions under which the component services can be exploited. Gartner analysts consider Context Delivery Architecture (CoDA) as the next step in the evolution of Service-Oriented Architecture; see [12–14]. In CoDA the functioning of software elements (services or event handlers) is determined not only by the input to the element, but also by the secondary sources of information – the context; two invocations of the same service with the same parameters may yield different results in different circumstances, i.e. within different contexts.

To reflect the varying nature of context and to ensure a universal applicability of context-aware systems, context is typically represented at different levels of abstraction [15]. At the first level of raw context sources there are context data coming from sensor devices, or user applications. At the next levels, context is represented using abstraction approaches of varying complexity. The work in [16] reviews models of context that range from key-value models, to mark-up schemes, graphical models, object-oriented models, logic-based models and ontology-based models. In [17] an ontological model of the W4H classification for context was proposed. The W4H ontology provides a set of general classes, properties, and relations exploiting the five semantic dimensions: identity (who), location (where), time (when), activity (what) and device profiles (how). The five dimensions of context have been also pointed out earlier in [18] where it was stated that context should include the 'five W': Who, What, Where, When, and Why. For example, by 'Who', they mean that it is not enough to identify a person as a customer; the person's past actions and service related background should also be identified for better service provision. 'What' refers to the activities conducted by the people involved in the context and interactions between them. 'Where' represents location data. 'When' is related to time. 'Why' specifies the reason for 'Who' did 'What'. 'Why' represents a complicated notion and acts as the driving force for context sensitive information systems.

An important aspect in the design of context-aware applications concerns modelling languages, which take context explicitly into account. The first such effort was ContextUML a UML-based modeling language which was specifically designed for context-aware Web service development and applies model-driven development principles; see [19]. In a Web-service-based environment ContextUML considers that context contains any information that can be used by a Web service to adjust its execution and output. Examples of contexts in ContextUML are: (i) contexts related to a service requester (mostly it is the client who invokes a service), including the requester's identification information, personal preferences, current situation (e.g., location), and other information (e.g., friends list, calendar); (ii) contexts related to a Web service, such as service location, service status (e.g., available, busy), and its QoS attributes (e.g., price, reliability); and (iii) other contexts like time and weather information. ContextUML has been adopted for the development of a model-driven platform, called ContextServ, which is used to develop context-aware Web applications; see [6]. There exist modelling

efforts that attempt to treat service logic and context handling as separate concerns: the first is the work in [20], where they modified ContextUML using Aspect-Oriented Programming (AOP) principles the second is the work in [21] that leveraged ideas from model driven development (MDD) and AOP in order to define a conceptual context model and then mapped these ideas to a UML framework.

A problem that arises in context detection is related to imperfect observations (e.g., sensor readings) that lead to the estimation of the current user situation. Many researchers (see [22–24]) suggested filtering or repairing problematic contexts (e.g., inaccurate, incomplete or noisy contexts). For example, to deal with this shortcoming, Anagnostoloulos et al. in [25] propose the use of Fuzzy Logic theory with the purpose of determining (inferring) and reasoning about the current situation of the involved user. In this approach, captured, imperfect contextual information is matched against pre-developed situation ontologies in order to approximately infer the current user context. In [24] a hybrid approach is proposed in order to detect problematic contexts and resolve resulting context inconsistencies with the help of context-aware application semantics.

Our work focuses on detecting context changes which correspond to either atomic or complex events and use complex event processing to model and identify them. Similarly to [26], we focus on events as a source of context because they are snippets of the past activities; therefore event processing may be viewed as a context detecting technology. Event processing results may be transferred to other applications, injecting context related information into services and processes. Based on the context definition of Dey and Abowd [1] and the associated five dimensions of context expressed in ontological model of the W4H [17], we define a high-level context model following an object-based modelling approach which can be easily specialized for different applications. We use semantic querying to extract contextual information from event payloads. Moreover, we exploit the reasoning capabilities of Situation- Action- Networks to enable dynamic derivation of context from multiple event streams and external services.

3 Context Model

We propose a context model as a stepping stone for facilitating event-based context detection and derivation functionality, in order to better understand situations in dynamic service oriented environments that demand for new additional information sources or/and lead to a number of service adaptations as means for successfully coping with dynamic environmental changes. In order to achieve the goal of extracting contextual information, analyzing them and then deriving higher level context, we follow an event-based context modelling approach. In this section, we present such a Context Model (Fig. 2), expressed in UML 2.0 class diagram. This model is based on the W4H model [17] that describes the five main elements associated within a context; the five elements are arranged into a quintuple (When, What, Where, Who, How).

This Context Model expresses the temporal (i.e. When), spatial (i.e. Where), declarative (i.e. Who, What) and explanatory (i.e. How) dimensions of context having as central point of focus the notion of Entity. We refer to either physical or virtual entities with specific profiles and preferences that characterise them (e.g. vessel, port authority information system etc.). This way context obtains substance around the notion of an

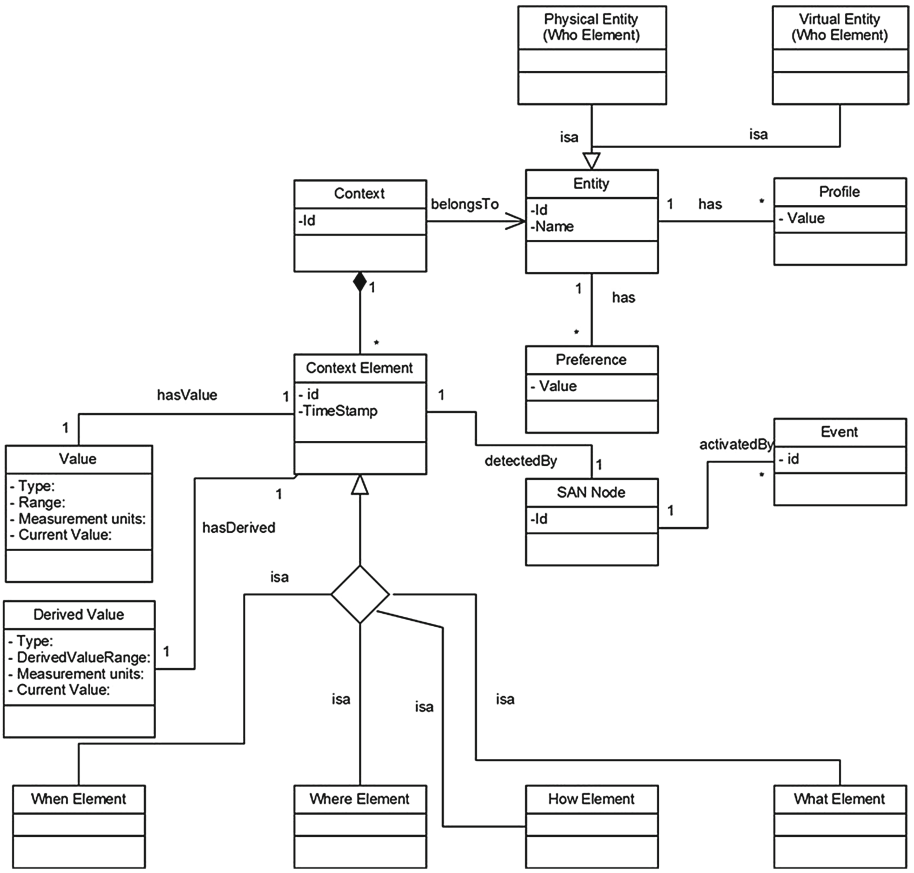

Fig. 2. Context Model

entity which can be a customer of an event marketplace system. The context class in our model constitutes the aggregation of several different context elements that may refer to five dimensions of context. Each Context element can have a value that can be acquired from the situation node of a SAN and/or a derived value that arises from any kind of reasoning process or call of external services. All context related information can be captured as objects which can store either a single scalar value or multiple values such as vectors, sets, lists etc. As any of the available context models [8], our model needs to become domain or application specific in order to be useful. Next, we show how SAN Editor can be used to specialize and instantiate the generic context model.

4 Event-Based Context Management

In this section we discuss our approach for both detecting and deriving contextual information based on events. Specifically, the context detection refers to a mechanism for querying events for updating contextual information while the context derivation

involves the acquisition of higher level context compared to the lower level information that events carry. Both approaches use SANs expressed in a RDF-based language that is presented in the second subsection.

4.1 Detecting and Deriving Context from Events

In our context modelling approach and implementation, we consider entities as being able to own SAN trees. The scope of context elements is distinguished into three levels:

- "Local": Context elements can be updated and used only by a specific SAN instance, e.g., wind velocity value of a specific area can be updated only by the sensors of a specific vessel's event stream that is handled by a specific SAN.
- "Entity": Context elements can be updated and used by any of the SANs owned by the same entity, e.g., wind velocity value of a specific area can be updated by the sensors of different vessels' event streams that are handled by more than one SANs owned by a Port Authority Information System (i.e. entity that owns SANs).
- "Global": Context elements can be updated and used by all SANs independently to which entity they belong to, e.g., wind velocity value of a specific area can be updated by the sensors of different vessels' event streams but also from a National Meteorological Service (i.e. different entities).

Using the SAN Editor, i.e. a dedicated graphical tool for designing SANs, we can perform context model specializations based on the application scenario and can formulate the necessary queries to events for extracting contextual information. Using this editor, SANs are visualised in two different tree-like graphical representations and can be exported in an executable format, i.e. SAN language (presented below). We provide two approaches for acquiring context from simple or complex events and instantiating our context model. Both approaches use the SAN Editor for:

1. defining SPARQL queries to specific RDF event payload information that can update the values of an entity's context elements; and
2. defining SANs that can use information from several event streams, analyse them and/or combine them with external services, in order to update the derived values of context elements. In this way, we succeed in acquiring higher level context compared to the lower level information that events carry.

Regardless the approach used for detecting context, all contextual information is stored in a dedicated Context Repository. We provide an API for the use of different types of repositories. We have tested our implementation using a relational database and an in-memory (implemented in Java) context-repository for the purpose of our marine related experiments.

The application of both approaches is presented in the following Sect. 5 through the marine vessel traffic illustrative scenario, which uses events related to marine traffic control that can be used to detect potentially dangerous vessel movements informing a controller when two vessels are approaching each other.

4.2 SAN RDF-Based Language

SAN definitions, generated by SAN editor, are stored in files using an RDF-based language. Currently the RDF/N3 format is supported by SAN engine but more RDF formats can be added in the future.

A set of RDF classes have been defined and have special meaning to SAN engine. These classes describe the SAN node types, such as Goals, Situations, Actions, or describe their properties, for instance name, subsequent nodes or actions, complex event patterns (CEPAT) expressions. These SAN definitions must have the following outline:

- Prefix specifications– Prefixes are used as short names of namespace URIs in order to define the domain (and origin) of the various RDF elements
- one or more SAN entity specification blocks
  - one entity specification, referencing the SANs (Root Goals) contained in that particular entity
  - any number of SANs specification blocks
    - Root Goal specification block
    - Situation specification block
      - Defines interesting/critical Situations as Complex Event Patterns (CEPATS). CEPATS are expressed using EP-SPARQL [27]
    - Context Condition specification block
    - Action/Subgoal specification block
      - Primitive Action specification block
      - Complex Action specification block - Sequence Action, Selector Action, Parallel Any/All/Timeout specification blocks
      - Subgoal specification block - Same as Root Goal (without auto-start or other root specific features)

In the following Table 1 we present a part (due to page limitations) of the SAN language specification expressed in Backus Normal Form (BNF). This formal language description specifies the main concepts of SAN language, i.e. Root Goal, Goal, Situation, Context Condition and Action. The Decorators carry behavioural specifications on how the subordinate actions or goals should be executed.

In Table 2, we present a SAN language example using a simplified excerpt of the marine vessel related SAN. In this example the entity “Port Authority” owns a SAN that undertakes the task of monitoring the safe sailing of a high speed vessel.

5 Illustrative Scenario

A vast amount of real time events are available from portals connected to automatic identification systems (AIS) that contain important vessel information worldwide (e.g., speed, course, vessel type, wind conditions etc.) and the several different users/authorities that might be interested in them. In order to exploit efficiently all these information in an automated way we use our context model and present how it can be specialized for the specific application domain while we give a glimpse to its possible

Table 1. SAN language specifications in BNF.

```
# SAN SPECIFICATIONS
<NODE_NAME> ::= ";" "san:name" <STRING>
<ROOT GOAL> ::= <URI> "a" "san:RootGoal"
                <NODE_NAME>
                <GOAL_DETAILS>
<GOAL> ::= <URI> "a" "san:Goal"
                <NODE_NAME>
                <GOAL_DETAILS>
<GOAL_DETAILS> ::= ";" "san:hasSituation" <URI>
                    ";" "san:hasContextCondition" <URI>
                    ";" "san:hasAction" <URI>
                    "."
<SITUATION> ::= <URI> "a" "san:Situation"
                <NODE_NAME>
                ";" "san:dialect" " 'EP-SPARQL' "
                ";" "san:defined-by" <EP-SPARQL-QUERY>
                "."
<CONTEXT_CONDITION> ::= <URI> "a" "san:Situation"
                        <NODE_NAME>
                        ";" "san:dialect" <STRING>
                        ";" "san:defined-by" <STRING>
                        "."

# ACTION SPECIFICATIONS
<ACTION> ::= <PRIMITIVE_ACTION> | <ABSTRACT_ACTION> | <CALCULATION_ACTION> |
<MOUNT_ACTION> | <PARALLEL_ANY_ACTION> | <PARALLEL_ALL_ACTION> |
<SEQUENCE_ACTION> | <SELECTOR_ACTION>

# DECORATOR SPECIFICATIONS
<DECORATOR> ::= <LOOP_DECORATOR> | <COUNTER_DECORATOR> |
<TIMER_DECORATOR> | <SUCCESS_DECORATOR> | <FAILURE_DECORATOR> |
<PRINT_DECORATOR> | <BREAK_DECORATOR>
```

run time instantiations. For the purposes of our illustrative scenario we have used the AIS Hub portal (http://www.aishub.net/) that shares such vessel tracking information. These real-time data are fed into our system as RDF events, through an appropriate adapter that we have developed.

Context Model Specialization: Our context model needs to become application specific in order to be useful. We focus on context model specialisation which pertains the definition of entities along with their context elements necessary for capturing the context in terms of a specific application scenario. We use the marine vessel traffic scenario which is related to vessel and marine traffic control observing systems.

In this scenario, we consider the entity Port Authority as the owner of all SANs discussed below while the entity of interest is the Vessel. In order to capture contextual information related to Vessels' context, we have defined the following Context Elements (using SANs) that shape the specialization of our context model: Speed, Course, Position, Status, Distance2Port. In Fig. 3, a screenshot of the SAN Editor is presented that depicts this context model specialization.

Table 2. SAN language example.

Definition	Example
Entity	:_Port_Authority a san:SANEntity ; san:hasRootGoal :_Highspeed_Vessel_Safe_Sailing ; san:auto-start "yes"^^xsd:string ; san:name "Port Authority"^^xsd:string .
Root Goal	: _Highspeed_Vessel_Safe_Sailing a san:RootGoal ; san:hasSituation : _Vessel_Moved ; san:hasContextCondition :_Check_vessel_speed ; san:hasAction :_Warn_Vessel ; san:auto-start "yes"^^xsd:string ; san:name "Highspeed Vessel Safe Sailing"^^xsd:string .
Situation	:_Vessel_Moved a san:Situation ; san:dialect "default"^^xsd:string ; san:defined-by " http://streams.event-processing.org/ids/ VesselStream s "^^xsd:string ; san:hasContextualizer : _Vessel_Moved_Contextualizer; san:name "A 'Vessel Moved' Event received"^^xsd:string . :_Vessel_Moved_Contextualizer a san:Contextualizer ; san:type "RDF"^^xsd:string ; san:contextualizer-query :_Vessel_Moved_Cntxlzr_Qry1 . : _Vessel_Moved_Cntxlzr_Qry1 a san:ContextualizerQuery ; san:language "SPARQL"^^xsd:string ; san:context "LOCAL"^^xsd:string ; san:dialect "default"^^xsd:string ; san:defined-by " SELECT ?vessel_id ?name ?speed WHERE { ?uri :MMSI ?vessel_id . ?uri :Name ?name . ?uri :Speed ?speed } "^^xsd:string .

Context Condition	:_Check_vessel_speed a san:ContextCondition ;
	san:dialect "JS"^^xsd:string ;
	san:defined-by "/* JS code */
	// code executed right away.
	var id = ctx.getItem('vessel_id')[0];
	var speed = ctx.getItem('speed')[0];
	// a function definition (not-executed)
	function check(speed) {
	// check if speed over 50 knots
	return (speed > 50); }
	// calling a function
	check(); // the result of the last statement is also
	// the result of the Context Cond. node.
	// i.e. true or false in this case.
	"^^xsd:string ;
	san:name "Check Vessel Speed"^^xsd:string .

Primitive Action	: _Warn_Vessel a san:PrimitiveAction ;
	san:command "EXPRESSION"^^xsd:string ;
	san:dialect "JS"^^xsd:string ;
	san:defined-by "/* JS code */
	var name = ctx.getItem('name')[0];
	// write a message to console
	out.println('Contact vessel '+name+
	' and ask to slow down');
	"^^xsd:string ;
	san:name "Warn vessel"^^xsd:string .

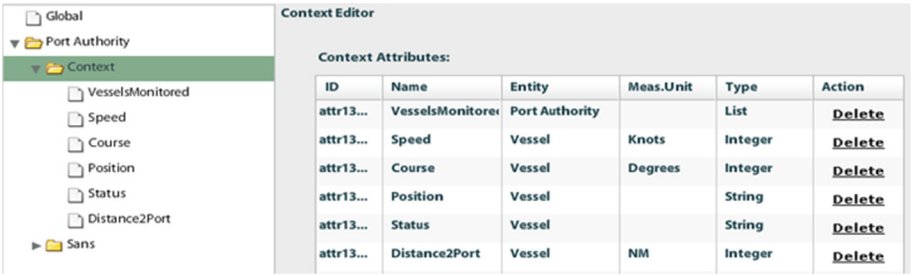

Fig. 3. Context Model Specialisation using SAN Editor

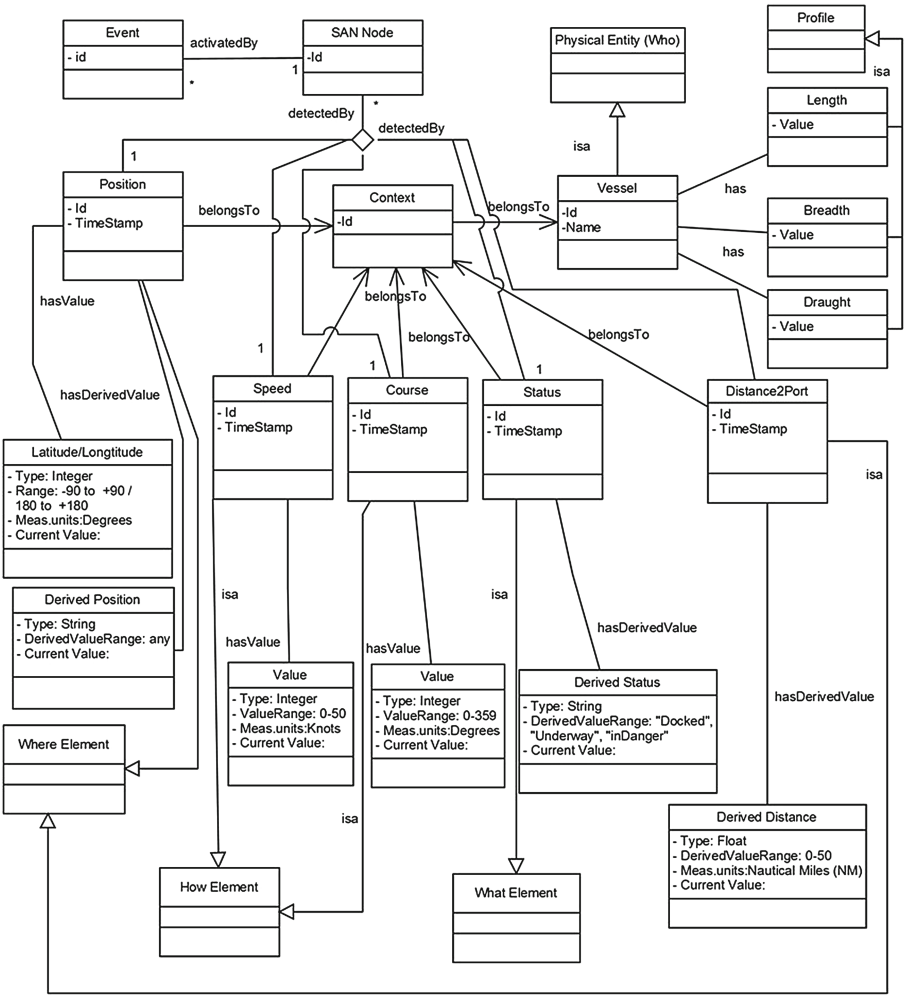

Fig. 4. Context Model Specialisation for the marine vessel traffic scenario

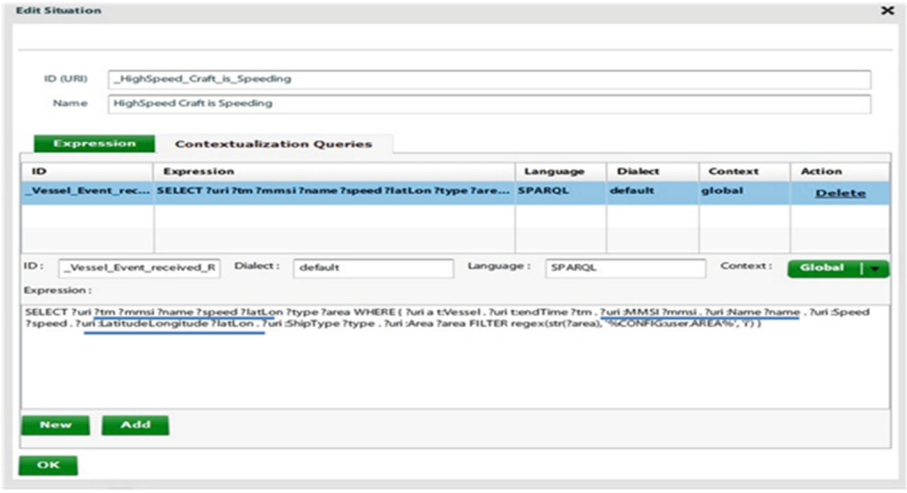

Fig. 5 SAN Editor screenshot – performing SPARQL queries (about position)

In Fig. 4, the reader can find the complete list of the five context elements associated with the Vessel entity, specialising the context model for the marine vessel traffic scenario. This model specialisation will be instantiated at run time through the context detection and derivation approaches that are presented below.

Detecting Context: In this section, we discuss our first approach for acquiring context from simple or complex events and instantiating our context model. Using SAN Editor, we are able to define SPARQL queries to specific event payload information that update the values of an entity's context elements. During our experiment we received events regarding a specific vessel called "Risoluto". Details regarding the entity such as profile information automatically update the context of this entity based on the detected events in the situation node of a SAN. Figure 5 depicts a screenshot of SAN editor with the required SPARQL queries for instantiating the "Position" context element of the vessel entity (Latitude/Longitude). Specifically, we query the vessel entity event payload with respect to the "LatLon" information. Similarly, other queries are used in the editor regarding the "Speed" and "Course" context elements and refer to event-based detection of low level context.

Deriving Context using SANs: this refers to our second approach that we apply for extracting context from simple or complex events and instantiating our context model using SANs. We define a number of SANs that can use information from several event streams and combine them with external services in order to update the derived value class of context elements. In this way, we succeed in acquiring higher level context compared to the lower level information that events carry.

This context derivation can be complex and may involve multi-level SANs. Figures 1 and 6 show a SAN that upon traversal will be able to update the derived value class of the Status context element. Specifically, the status of the vessel becomes "Docked" whenever we detect a vessel that has been stopped and its distance from any port is close to zero or "UnderWay" whenever vessel's speed is close to average and

"In Danger" when the system realizes that the vessel has almost stopped (away from any port) and strong winds are blowing from the side. The run-time execution of the specific SAN led to the derivation of a number of vessels' statuses. A pop up alert has been added in order to better demonstrate the context derivation regarding the Status context element for each vessel. Such contextual information can provide the basis for intelligent services in today's event marketplaces that will provide context-aware value added functionalities (e.g. dynamic event subscriptions, service/workflow adaptation).

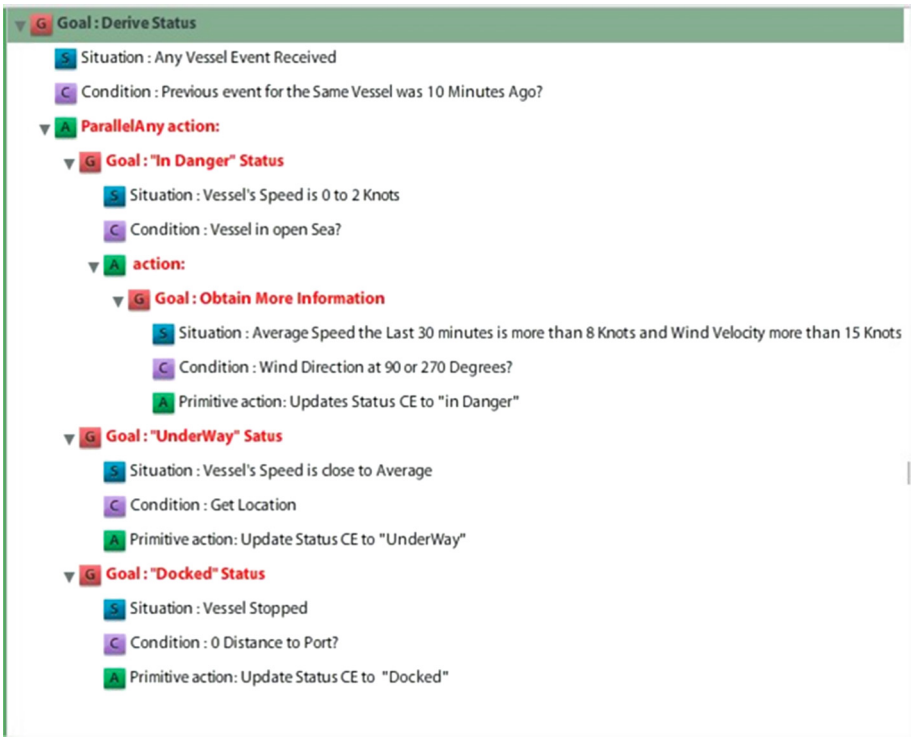

Fig. 6. SAN for the marine vessel traffic scenario

6 Conclusions

In this paper we presented methods and tools for enhancing context detection and management based on events. This proposed context management approach presented here is considered appropriate for the needs of event marketplaces. We described a Context Model that was used by the developed mechanisms for performing event-based context detection and presented two different approaches for detecting low level context (using SAN Editor) and deriving higher-level contextual information using Situation-Action-Networks (SANs). We provided with a meaningful context model specialization and demonstrated how simple or complex events coming from an event marketplace can be used and combined with external services, in order to derive higher

level context with the use of SANs. For this demonstration we used real-time data from the AIS Hub portal that were published as RDF events into our system, using the appropriate event adapter.

This proposed context management approach presented here is considered appropriate for the needs of value added services that can be developed in order to provide intelligent and cost efficient characteristics in today's event marketplaces. Such services can include dynamic event subscriptions recommenders that propose the appropriate times for subscribing and unsubscribing to heterogeneous event sources [28] or service adaptation frameworks that can detect and implement meaningful adaptations to business processes [3] based on situations and context-related information.

Acknowledgment. This work has been partly funded by the European Commission within the FP7 projects PLAY (258659) and REFLEX (63756). The authors would like to thank the project partners for their advices and comments regarding this work.

References

1. Dey, A.K., Abowd, G.D.: Towards a better understanding of context and context-awareness. In: Proceedings of the PrCHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness, pp. 304–307 (2000)
2. Stühmer, S., Stojanovic, N.: Large-scale, situation-driven and quality-aware event marketplace: the concept, challenges and opportunities. In: Proceedings of the 5th ACM International Conference on Distributed Event-Based System, NY, USA, pp. 403–404 (2011)
3. Verginadis, Y., Patiniotakis, I., Papageorgiou, N., Stuehmer, R.: Service adaptation recommender in the event marketplace: conceptual view. In: Garcia-Castro, R., Fensel, D., Antoniou, G. (eds.) ESWC 2011. LNCS, vol. 7117, pp. 194–201. Springer, Heidelberg (2012)
4. Abowd, G.D., Ebling, M., Hunt, G., Lei, H., Gellersen, H.W.: Context-aware computing. IEEE Pervasive Comput. **1**(3), 22–23 (2002)
5. Sheng, Q.Z., Nambiar, U., Sheth, A.P., Srivastava, B., Maamar, Z., Elnaffar, S.: WS3: international workshop on context-enabled source and service selection, integration and adaptation. In: Proceedings of the International Workshop on Context Enabled Source and Service Selection, Integration and Adaptation, Beijing, China, pp. 1263–1264 (2008)
6. Sheng, Q.Z., Yu, J., Dustdar, S.: Enabling Context-Aware Web Services: Methods, Architectures, and Technologies, 1st edn. Chapman and Hall/CRC, Boca Raton (2010)
7. Villegas, N.M., Müller, H.A.: Managing dynamic context to optimize smart interactions and services. In: Chignell, M., Cordy, J., Ng, J., Yesha, Y. (eds.) The Smart Internet. LNCS, vol. 6400, pp. 289–318. Springer, Heidelberg (2010)
8. Truong, H.L., Dustdar, S.: A survey on context-aware web service systems. Int. J. Web Inform. Syst. **5**(1), 5–31 (2009)
9. Hong, J., Suh, E., Kim, S.J.: Context-aware systems: a literature review and classification. Expert Syst. Appl. **36**(4), 8509–8522 (2009)
10. Kapitsaki, G.M., Prezerakos, G.N., Tselikas, N.D., Venieris, I.S.: Context-aware service engineering: a survey. J. Syst. Softw. **82**(8), 1285–1297 (2009)
11. Bucchiarone, A., Cappiello, C., di Nitto, E., Kazhamiakin, R., Mazza, V., Pistore, M.: A context-driven adaptation process for service-based applications. In: Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems, pp. 50–56 (2010)

12. Natis, Y.V., Clark, W., Valdes, R.: Context delivery architecture: putting SOA in context. Gartner Research, ID Number: G00152306 (2007)
13. Clark, W., Lapkin, A.: Fundamentals of context delivery architecture: introduction and definitions. Gartner Research ID Number: G00161876 (2008)
14. Clark, W.: Fundamentals of context delivery architecture: provisioning context-enriched services. Gartner Research, ID Number: G00200649 (2010)
15. Luther, M., Fukazawa, Y., Wagner, M., Kurakake, S.: Situational reasoning for task-oriented mobile service recommendation. Knowl. Eng. Rev. **23**(01), 7–19 (2008)
16. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive Mob. Comput. **6**(2), 161–180 (2010)
17. Truong, H.L., Manzoor, A., Dustdar, S.: On modeling, collecting and utilizing context information for disaster responses in pervasive environments. In: Proceedings of the 1st International Workshop on Context-Aware Software Technology and Applications, pp. 25–28 (2009)
18. Abowd, G.D., Mynatt, E.D.: Charting past, present, and future research in ubiquitous computing. ACM Trans. Comput. Hum. Interact. **7**(1), 29–58 (2000)
19. Sheng, Q., Benatallah, B.: ContextUML: a UML-based modeling language for model-driven development of context-aware web services development. In: Proceedings of the International Conference on Mobile Business (ICMB 2005), Sydney, Australia, pp. 206–212 (2005)
20. Prezerakos, G.N., Tselikas, N.D., Cortese, G.: Model-driven composition of context-aware web services using ContextUML and aspects. In: Proceedings of the IEEE International Conference on Web Services, (ICWS), pp. 320–329 (2007)
21. Grassi, V., Sindico, A.: Towards model driven design of service-based context-aware applications. In: International Workshop on Engineering of Software Services for Pervasive Environments: in Conjunction with the 6th ESEC/FSE Joint Meeting, pp. 69–74 (2007)
22. Bu, Y., Gu, T., Tao, X., Li, J., Chen, S., Lu, J.: Managing quality of context in pervasive computing. In: QSIC 2006, pp. 193–200. IEEE Computer Society, Washington, DC (2006)
23. Xu, C., Cheung, S.C., Chan, W.K., Ye, C.: Heuristics-based strategies for resolving context inconsistencies in pervasive computing applications. In: ICDCS 2008, pp. 713–721. IEEE Computer Society, Washington, DC (2008)
24. Chen, C., Ye, C., Jacobsen, H.-A.: Hybrid context inconsistency resolution for context-aware services. In: Proceedings of 9th IEEE International Conference on Pervasive Computing and Communications, PerCom 2011 (2011)
25. Anagnostopoulos, C., Ntarladimas, Y., Hadjiefthymiades, S.: Situational computing: an innovative architecture with imprecise reasoning. J. Syst. Softw. **80**(12), 1993–2014 (2007)
26. Etzion, O., Skarbovsky, I., Magid, Y., Zolotorevsky, N., Rabinovich, E.: Context aware computing and its utilization in event-based systems. Tutorial presented in DEBS, Cambridge, UK (2010)
27. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: WWW 2011, pp. 635–644 (2011)
28. Verginadis, Y., Patiniotakis, I., Papageorgiou, N., Apostolou, D., Mentzas, G.: A goal driven dynamic event subscription approach. In: The 6th ACM International Conference on Distributed Event-Based Systems (DEBS 2012), Berlin, Germany (2012)