SensorStream: A Semantic Real-Time Stream Management System

Dimitrios-Emmanuel Spanos*, Periklis Stavrou, Nikolas Mitrou

School of Electrical and Computer Engineering,
National Technical University of Athens, Zografou 157 73, Athens, Greece
E-mail: {dspanos, pstavrou}@cn.ntua.gr,mitrou@cs.ntua.gr
*Corresponding author

Nikolaos Konstantinou

Athens Information Technology
19,5km Markopoulo Ave., Peania, Greece
E-mail: nkons@ait.edu.gr

Abstract: As data proliferates at increasing rates, the need for real-time stream processing applications increases as well. In the same way that data stream management systems have emerged from the database community, there is now a similar concern in managing dynamic knowledge among the Semantic Web community. Unfortunately, early relevant approaches are to a large extent theoretical and do not present convincing evidence of their efficiency in real dynamic environments. In this paper, we present a framework for the effective, real-time processing of streaming data and we define and analyze in depth its key components. Our framework serves as a basis for the implementation of the SensorStream prototype, on which we run numerous performance and scalability measurements that outline its behaviour and demonstrate its suitability and scalability for solutions that require real-time information processing from distributed and heterogeneous data sources.

Keywords: Stream data; sensors; semantic management; data management; ontology; dynamic knowledge; scalable framework; real-time; windowing; information processing; distributed sources; heterogeneity; performance measurements.

Reference to this paper should be made as follows: Spanos, D.E., Stavrou, P., Konstantinou, N. and Mitrou, N. (2011) 'SensorStream: A Semantic Real-Time Stream Management System', *Int. J. of Ad Hoc and Ubiquitous Computing*, Vol. x, Nos. x/x, pp.xxx–xxx.

Biographical notes: Dimitrios-Emmanuel Spanos received his diploma degree in Electrical and Computer Engineering from the National Technical University of Athens (NTUA) in 2005 and a Master in Business Administration (MBA) from the Athens University of Economics and Business (AUEB) in 2007. Since 2007, he is a PhD candidate in NTUA and a member of the Multimedia Communications and Web Technologies Research Group. His research interests lie in the field of Semantic Web and knowledge representation including, among others, relational database to ontology mapping, semantic annotation of multimedia content from sensor networks and semantic search in the context of libraries and open access repositories. He is a member of the Technical Chamber of Greece.

Periklis Stavrou graduated in 2008 from National Technical University of Athens (NTUA) with a Dipl-Ing degree in Electrical and Computer Engineering. Since October 2008 he is part of the Multimedia Communications and Web Technologies Research Group in NTUA, working towards his PhD. Among his main research interests are semantic annotation, data fusion using semantic web technologies and knowledge representation and reasoning in distributed systems. He is a member of the Technical Chamber of Greece.

Nikolaos Konstantinou received his Engineer Diploma from the National Technical University of Athens (NTUA), School of Electrical and Computer Engineering (ECE), in February 2004. In November 2004, after having fulfilled his military obligations he was accepted as a Ph.D. candidate in NTUA, School of ECE, Division of Communication, Electronic and Information Engineering. In December 2009 he successfully defended his Ph.D. thesis, entitled "Semantic information enrichment in relational databases and context-aware systems". During his studies, Dr. Konstantinou participated in a number of nationally-funded research programs while also obtaining valuable experience working as a software engineer. In February 2010 he joined the Athens Information Technology where he currently works as a post-doctoral researcher.

His research interests include knowledge management, context-sensitive systems and information flow in the internet of things. His research resulted in several publications, with a very encouraging reception in terms of citations. He is a member of the Technical Chamber of Greece and a member of the IEEE.

Nikolas Mitrou is a Professor in National Technical University of Athens. Prof. Mitrou's research interests are in the areas of digital communication systems & networks (broadband networks in particular) and networked multimedia in all range of studies: design, implementation, modelling, performance evaluation and optimization. Since 1988 Prof. Mitrou has been actively involved in many RACE, ACTS and ESPRIT projects and he was the coordinator of one of them (AC235, WATT). Prof. Mitrou is a member of the IEEE, member of the IFIP WG 6.3 and member of the Technical Chamber of Greece.

1 Introduction

In recent years, technological advancements in microsensor technology and increasing business needs have laid the ground for an emerging class of applications that shift their focus toward data streams. The availability of large scale sensor deployments results in huge, often unbounded, quantities of data generated by all kinds of sensor devices, calling for efficient management and storage. In the same time, businesses seek ways to perform analysis of their data, be it financial, medical, customer or corporate network data, as soon as it becomes available, in order to have an accurate picture of the market situation and be able to respond in a timely manner. These needs originally led to the development of several Data Stream Management Systems (DSMS) during the last decade (Arasu et al., 2004; Chandrasekaran et al., 2003; Abadi et al., 2005), so as to fill the gap left by traditional Database Management Systems, which cannot deal with continuous, real-time sequences of data. DSMS introduced a novel rationale that is not based on persistent storage of all available data and user-invoked queries, but instead advocates on the fly stream manipulation and permanent monitoring queries.

In the parallel universe of Knowledge Engineering, we can see that while reasoners have excelled their performance with regard to static knowledge, the issue of reasoning with frequently changing facts and assertions has not been adequately addressed yet. *Stream reasoning* has been coined as a term that describes the above problem (Della Valle et al., 2009) of performing reasoning on a knowledge base comprising stable or occasionally changing terminological axioms and a stream of incoming assertions or facts. Advances and solutions to this problem will progressively lead the way for smarter and more complex applications, including traffic management and fastest route planning, environmental monitoring, surveillance and object tracking scenarios, and disease outburst detection applications among others.

So far, work in this relatively new field includes re-definition of the RDF model and enhancements of current W3C recommendations for time-aware knowledge representation and querying (Gutierrez et al., 2007; Pugliese et al., 2008; Rodriguez et al., 2009; Bolles et al., 2008; Barbieri et al., 2009), incremental reasoning techniques (Cuenca Grau et al., 2007; Parsia et al., 2006; Volz et al., 2005) and frameworks that use combined experience from the database and Semantic Web research fields (Walavalkar et al., 2008; Barbieri et al., 2010). However, despite the need for working applications that can manage and query efficiently streaming data, there is considerable lack of such tools in the literature since most approaches remain theoretical and fail to provide evaluation of their performance that will serve as adequate proof of their utility.

In this paper, we present a generic ontology-based architecture for the efficient SPARQL querying of streaming XML data and event recognition. Our framework does not impose any restriction to the nature of the incoming data, as long as they are encoded in a well-formed XML template. Moreover, our framework can be tailored to any application context according to the ontology selected for use. Sets of domain-specific rules define the system's attitude and the actions to be taken, when a high-level event is being recognized. We created the SensorStream prototype on top of Jena Semantic Web framework (Carroll et al., 2004) as a working implementation of our architecture and ran experiments to assess its performance. Results show that our system is perfectly scalable, has near real-time response and can handle changes in the terminological part of the ontology, providing sound inferences at all times.

The rest of the paper is organized as follows: in Section 2, some background information on the stream management problem as well as the motivation behind our work are presented. Section 3 mentions the related work, while Section 4 presents in detail our framework. Section 5 provides an evaluation of the efficiency of our system and the simulation experiments run on. Finally, Section 6 highlights the conclusions and sketches out future work.

2 Background and Motivation

In general, the issue of efficient management and querying of dynamic data is related to several research domains. We have already mentioned that the need for stream processing had originally arisen within the database community when it became evident

that typical DBMS could not support applications where data is dynamic and processing requests are event-driven rather than human-initiated. Traditional Database Management Systems (DBMS) are in effect passive repositories of data that execute queries on user's demand, therefore they are inadequate to handle streaming data of possibly infinite size and execute continuous queries on them. This has led to the introduction of several prototype systems (Arasu et al., 2004; Chandrasekaran et al., 2003; Abadi et al., 2005; Chen et al., 2000; Liu et al., 1999) that constitute the groundwork of the recently emerged stream management problem (Babcock et al., 2002).

In a nutshell, DSMS introduce various methods in order to overcome the unbounded nature of stream data and the problems it poses on query execution. Most DSMS extend the relational model by including a timestamp as a label for each relational tuple. This timestamp can either denote the arrival time of the tuple in the system or constitute a property of the tuple itself, e.g. the occurrence time of the event this tuple describes. The first kind of timestamp is called *implicit*, while the second one is referred to as *explicit*. The timestamp is usually the indicator value acting as a basis for the formulation of stream data *windows*. The notion of windows in stream manipulation tasks is a common one, since it greatly simplifies query execution by considering only finite subsets of the entire stream. A window over a stream is defined as a finite portion of the stream at some point in time. This portion can be defined in terms of either time or number of tuples: in the former case, the window is said to be *time-based* or *logical*, while in the latter case, the window is called *tuple-based* or *physical*. The elements of the stream that form the window change as new stream data becomes available in a DSMS system.

According to the definitions given in (Patroumpas and Sellis, 2006), a *logical window* for a stream $\mathbb{S}$ of tuples is the set of tuples with timestamps occurring in the $[t_1, t_2]$ interval:

$$w_l(\mathbb{S}, t_1, t_2) = \{\mathbb{S}(\tau), t_1 \leq \tau \leq t_2\} \tag{1}$$

On the contrary, a *physical window* applied at time instant τ contains the N most recent tuples of a stream:

$$w_p(\mathbb{S}, N, \tau) = \{w_l(\mathbb{S}, t_1, \tau) : t_1 \leq \tau \wedge |w_l(\mathbb{S}, t_1, \tau)| \leq N \wedge \forall t_2 < t_1 : |w_l(\mathbb{S}, t_2, \tau)| > N\} \tag{2}$$

Intuitively, the above definition specifies t_1 as the timestamp of the oldest among the N most recent tuples and uses it as the start of a logical window spanning until τ, which denotes the current time instant.

Depending on the way that windows are updated, they can be categorized as *sliding*, *tumbling* or *landmark* windows. *Sliding windows* have a fixed length and both their bounds are moving at a predefined interval, replacing old stream elements with new ones. A sliding window with size w and progression step d can be defined as:

$$w_{sl}(\mathbb{S}, w, d, t_0, \tau) = \begin{cases} w_l(\mathbb{S}, t_0, \tau), & t_0 \leq \tau \leq t_0 + w \text{ and } mod(\tau - t_0, d) = 0 \\ w_l(\mathbb{S}, \tau - w, \tau), & t_0 + w < \tau \text{ and } mod(\tau - t_0, d) = 0 \\ w_{sl}(\mathbb{S}, w, d, t_0, \tau - dt), & mod(\tau - t_0, d) \neq 0 \end{cases} \tag{3}$$

where t_0 is the time of the initial application of the windowing method and τ is the current time instant. Equation (3) shows that sliding windows are computed at fixed time intervals d, starting at t_0. For intermediate times, the sliding window stays the same as the one computed in the preceding time instant (where dt denotes the time resolution of the system), hence the recursive nature of the last branch of the definition. Usually, progression step d is smaller than window size w, leading to overlaps between two successive sliding windows.

Tumbling windows can be considered as a special case of sliding windows, where the window is moving at a fixed interval of length equal to the window size ($d = w$). Thus, tumbling windows are pairwise disjoint and do not share any stream elements:

$$w_{tum}(\mathbb{S}, w, t_0, \tau) = w_{sl}(\mathbb{S}, w, w, t_0, \tau) \tag{4}$$

Landmark windows are another type of window, where the beginning bound t_l remains fixed while the other one is moving, resulting in a variable window size (Arasu et al., 2006):

$$w_{land}(\mathbb{S}, t_l, \tau) = \begin{cases} w_l(\mathbb{S}, t_l, \tau), & \tau \geq t_l \\ 0, & \tau < t_l \end{cases} \tag{5}$$

A graphical representation of the above three types of logical windows is shown in Figure 1. Physical (tuple-based) windows are defined analogously as their logical counterparts in Equations (3), (4) and (5). Besides the aforementioned rudimentary windowing methods, semantics of other windowing techniques have been proposed in the literature, such as the partitioned window technique (Li et al., 2005), where windows can be formed according to attributes of interest, and the much resembling notion of predicate windows (Ghanem et al., 2006). In our approach, we favour the use of a sliding physical window, which can be lightly formed and updated.

Other popular stream processing techniques include filtering, punctuation and synopses. Filtering, as the name suggests, mainly consists of applying criteria for the selection of a subset of stream elements and ignoring the rest, while punctuation uses special markers inside the stream to indicate that the stream elements that have arrived so far can be partially processed and blocking operators (e.g. a maximum or a sorting operator) can be applied to them. Synopses are mere summaries and aggregations of incoming data, extremely useful when approximate answers are required. A brief overview of these techniques is provided in (Maier et al., 2005).

Lately, a similar tendency has been observed in the Semantic Web research community. While the trade-off between expressive knowledge representation languages and reasoning complexity has been extensively documented so far (Baader et al., 2005) and tools that offer scalable and efficient ontology storage and reasoning solutions surface at an increasing rate (Broekstra et al., 2002; Carroll et al., 2004; Das and Srinivasan, 2009; Erling and Mikhailov, 2010), the focus has been mainly put on static collections of data and ontologies. Relatively few methods and frameworks have been proposed for the processing and management of dynamic semantic data and ontologies and even fewer working application examples have been implemented so far. Such efforts are presented in the next section.

3 Related Work

The lack of working stream processing applications in the Semantic Web context, pointed out in Section 2, is partly due to the fact that current popular W3C recommendations OWL 2 (Motik et al., 2009) and SPARQL (Prud'hommeaux and Seaborne, 2008) do not offer satisfactory support for time-evolving data. Modelling the notion of time is essential, since time is a primordial factor in dynamic systems and streaming applications. Considerable effort has been made towards expanding the RDF model so as to incorporate time. The most notable efforts to enhance RDF with time features and provide an adequate respective query language are presented in (Gutierrez et al., 2007; Pugliese et al., 2008). Another extension of RDF incorporating the notion of time is presented in (Rodriguez et al., 2009), where every RDF resource considered is time annotated. An extension of SPARQL is proposed along with the above Time-Annotated RDF extension and they are both applied in an indicative architecture to store and query time-varying data. Essentially, this extension works merely as a synopsis of a more complex RDF representation that uses few additional proprietary RDF predicates.

Related efforts that are worth mentioning are the Time Ontology in OWL (Hobbs and Pan, 2006), which describes general temporal information and O&M-OWL (Henson et al., 2009), modelling time series sensor observations. Likewise, additions to the SPARQL query language enabling continuous queries over windowed stream data have been incorporated in the C-SPARQL (Barbieri et al., 2009) and Streaming SPARQL (Bolles et al., 2008) approaches. As these approaches define new languages or update existing ones, they are significantly different from the work presented here.

Most of the approaches dealing with management and query processing of streaming data build on the combination of existing DSMS and experience on reasoning and storage techniques for static knowledge, while using some of the aforementioned time-enhanced languages. To name just one example, Walavalkar et al. (2008) build on existing mature projects Jena (Carroll et al., 2004) and TelegraphCQ (Chandrasekaran et al., 2003) for the reasoning procedure on the static ontology and stream processing respectively in order to diminish the processing time of RDF stream messages. The entire subsumption hierarchy for OWL classes and properties is computed only once in a preliminary stage, and stored in a database together with the inference results from domain and range statements. Then, on runtime, as RDF statements arrive, new statements are interpolated in the stream based on the inferences stored in the system's database. The performance of this approach clearly depends on the number of the initially inferred statements. The underlying assumption of this work is that all stream messages abide by a common static ontology schema, making no prevision for possible changes in the set of ontology axioms. Another drawback of this approach is the fact that it utilizes the stored inferences for only a pre-specified class of interest, ignoring stream messages not obeying this restriction.

A prototype architecture that provides a proof of concept for the efficiency of the C-SPARQL language in continuous querying of stream data is described in (Barbieri et al., 2010). This architecture combines a plain SPARQL engine with a DSMS to achieve the desired functionality. According to the above architecture, every continuous query is split into two parts: a static and a dynamic one, with the first one invoking the reasoner to infer new knowledge and the second one delegating execution to the underlying DSMS. This implies that reasoning takes place only once for every continuous query, right after its initial registration with the system. Thus, any modification in the terminological part of the knowledge base does not reflect in the results of a query that has already been registered.

Semantic Streams (Whitehouse et al., 2006) is a framework that specifies a predicate-based representation of sensors and their measurements and allows users to pose Prolog-like queries. The answers are evaluated using a backward chaining variant where a goal tree is constructed and consecutive variable bindings are tried out until an answer is found. The stream nature of sensor data seems to be overlooked in this approach and no performance tests are provided, though it is highly unlikely that real-time requirements are satisfied given that execution of Prolog programs can be highly inefficient. Another mainly theoretical framework for a stream processing middleware is DyKnow (Heintz et al., 2009), where emphasis is given on stream reasoning methods and on the devision of a representation language for policies and processes applied to the incoming streams.

SensorMasher (Le-Phuoc and Hauswirth, 2009) is a platform for publishing sensor data as Linked Data (Bizer et al., 2009), based on a layered architecture, similar to ours. However, the focus in that work lies on the exposure of sensor data to the user under different ways (Web service, HTTP representation, SPARQL

endpoint etc.) and less on the management aspect and real-time processing of the incoming stream data.

A context aware environmental monitoring system that relies on a set of rules and a reference ontology to deduce high-level events from low-level sensor readings is presented in (Calder et al., 2010). This approach shares several features with our system, nevertheless the streaming nature of incoming sensor data is neglected and the performance of the system in relevant dynamic environments remains unclear. Information extraction from data that arrives in streams is also performed by complex event processing architectures (Gyllstrom et al., 2007; Demers et al., 2007) that define proprietary query languages for data retrieval. These applications, though, are only marginally related to our work, since they lack semantic representation of both incoming data and application context. Our SensorStream system borrows ideas from several of the above approaches, building on previous work in context-aware middleware (Konstantinou et al., 2010).

4 The SensorStream Approach

In this section, we describe a layered architecture for the efficient real-time processing and reasoning with stream data. This architecture has served as a basis for the implementation of our SensorStream prototype and is depicted in Figure 2. In the rest of this section, we enumerate the core SensorStream components and detail on them.

Information sources. Information sources are the content providers for our system. In this paper, we give emphasis to streaming sources that constantly provide new, possibly unbounded, information. Such examples of information streams include sensor data, clickstream data, financial data or even network monitoring messages. Our framework is applicable to all kinds of streaming data, no matter what its source is. This suppleness is guaranteed by a common language all stream messages should adhere to, as explained later.

In some cases, an information source comprises a sensing device observing its environment and one or more distributed trackers. Trackers are software components that process data in order to extract more meaningful and useful information. For instance, a tracker can process live video streams in order to extract events of interest, such as in (Lien et al., 2007; Zhang and Chang, 2002), where the trackers extract events from baseball video streams, or in (Comaniciu and Meer, 1999; Bradski, 1998; Allen et al., 2001) where the algorithms deal with the face or object tracking problem.

The trackers are considered to be distributed, in the sense that there is no need of message exchange or communication between them at any level. A system like the hereby presented approach can rely on a number of distributed trackers without the need of a centralized coordination, with the exception of a common clock in cases when this is deemed necessary. In case the information processed needs to be consistent to a common clock and present continuity and timeliness, a time server could be contacted.

Message language and mechanism. The various system components need to communicate via well defined interfaces. Therefore, there is a need of specifying:

1. *A common language.* In the current case, the language of choice is XML, in order to allow interoperability with any specific tracker. The structure and content of XML messages can be conveniently tailored to the needs of any application by complying to specific schema definitions (XSDs). Thus, we relax the assumption made by the majority of approaches - as seen in Section 3 - where incoming stream data is a set of "ready to be consumed" RDF statements. This is clearly not the case for real applications and data generators, which usually do not provide data coupled with context or other semantic information.

   On the contrary, the assumption that data or measurements are readily available in XML is not a far-fetched one, especially in the case of financial or Web applications' data. In the case of sensing data, few developing effort is needed to convert the custom response format of a typical sensor device (also known as mote) to a XML template of choice. Thus, we can assume that such small-scale XML converters are part of every tracker's or data source's interface with the other framework components. This renders our framework directly applicable even to situations when regular motes, usually forming a wireless sensor network, are used as data providers supporting high level applications.

2. *Common or well interoperable technologies.* In order to establish communication inside a widely distributed network of heterogeneous devices, one has many options: the Java Messaging System (JMS), sockets or Web services to name a few. In the studied approach, Web services have been chosen because of the independence from technologies, portability to any platform, scalability and maturity of the recommended approaches. Apache Axis2, CXF, are only a few of the projects supporting the related W3C recommendations[1].

3. *Interfacing.* Well-defined and complete functionality should be exposed to the information sources and, more precisely, to the trackers. For the purpose of the current implementation, only one method is needed, in order to send the XML message carrying the information from the trackers to the server. Additional methods could be applied to send a server's response in the form

of commands to the trackers. Such interfacing methods are necessary when information sources need to receive feedback from the main processing unit and hence, adapt themselves in environmental conditions or intricate higher-level application needs, possibly expressed in the form of semantic rules.

Message fusion mechanism. In order to collect, separate and thereafter fuse the incoming messages, a wide variety of approaches exist in the literature. In fact, data fusion from a variety of sources constitutes a lively research domain with a variety of open issues and a highly active community (Liggins et al., 2009). As XML inherently carries only structural information, in order to combine information from multiple sources it is essential to enhance semantically this information and ensure semantic interoperability. In the approach presented here, mapping rules are used in order to combine information from multiple sources and leverage its meaning by producing higher level information.

To be more specific, the information contained in the XML stream messages contributes and modifies the knowledge of the system. Ideally, the intended behavior of the system consists of evolving the ontology representing the knowledge core of the system, according to the incoming events encoded in XML. To this end, a set of mapping rules are employed to add statements to the knowledge base. These rules are formed according to the following event-condition-action (ECA) pattern (Papamarkos et al., 2003):

on *event* if *condition* then *actions*

where the event in our case is a XML message arrival and the condition is applied to the incoming XML message. These mapping rules fetch data from the XML message and store it into the ontology model in the form of class individuals. They are expressed in a custom language allowing for combination of the XML and RDF technical spaces, where typical examples of conditions in the above clause include the presence of a specific XPath expression, whereas actions usually consist of the addition of individual assertions. These rules are constructed manually from a user who is aware of the possible templates of the incoming XML messages and the ontology structure as well. The full grammar specification of the mapping rule language and details on the system's user interfaces can be found in (Konstantinou et al., 2010).

This design choice automatically raises the limitation of a single common schema for all stream elements, assumed by other relevant approaches. Different XML templates can be considered for every distinct data generator, as long as there is at least one corresponding mapping rule to handle the ontology population. An alternative choice for the XML to OWL transformation would involve application of XSLT stylesheets to the incoming XML messages, following the GRDDL approach (Conolly, 2007). Although our implementation can handle either of these approaches, for the test scenario presented in this paper, we followed the mapping rules approach.

Information manipulation. With this term, we refer to information administering, processing, storing and exploiting. The technologies that are chosen have to be able to cooperate smoothly with each other. For instance, regarding storage, the available options include relational databases, distributed (NoSQL) databases, and semantic web RDF stores. Respectively, processing and querying should be done using SQL, keyword-based approaches[2], or SPARQL queries. Therefore, the decisions that have to be taken regarding the information flow and the related technologies are greatly affected by the purpose of the whole attempt.

In the present approach, we follow the Semantic Web point of view, in order to study the corresponding benefits and difficulties. However, such a view entails the necessity of making a number of additional choices.

1. *The reasoner.* We need to be aware that in the Semantic Web context, reasoning is an issue that needs to be specially taken care of. Poor decisions in the choice of a reasoner, selection of reasoner's connectivity method with the application and in some cases even fine tuning the reasoner can lead to serious performance degradation. A series of reasoners are nowadays available in the bibliography, including FaCT++ (Tsarkov and Horrocks, 2006), Pellet (Sirin et al., 2007), KAON2 (Motik et al., 2005) and OWLJessKB[3] among others. For the current implementation, the reasoner chosen is Pellet since it is the one that is open-source, Java-based, allows integration with the application's code, is based on a complete DL reasoning algorithm but, most importantly, Pellet has the feature of incremental reasoning, which is closely related to the characteristics a real-time system should present. The incremental classification feature of Pellet allows for improved performance in cases of frequent axiom modifications in the knowledge base, given that the classification hierarchy is incrementally updated instead of being recomputed from scratch with every axiom change. Furthermore, incremental consistency checking, ensures that in an environment of continuous $\mathcal{A}$Box additions or deletions, completion rules are applied to the updated completion graph and not starting again from scratch (Halaschek-Wiener et al., 2006).
2. *The ontology.* The ontology that has to be chosen in order to describe the concepts involved in each problem should be carefully selected while keeping in mind that a highly expressive ontology entails increased complexity and additional computational burden for the reasoning procedure. Thus, among the vast variety of ontologies available on the Web, special care needs to be taken in choosing one and tailoring it to the

application's needs. It is important to note that we do not consider a standard generic ontology that covers most application cases, although more than a few have been proposed in the related situation awareness literature, e.g. (Little and Rogova, 2009; Kokar et al., 2009). As the choice of the ontology to be used sets the application context, our framework is kept application-independent and domain-agnostic.

As we have seen in Section 3, the notion of time is vital in stream processing applications. Hence, we argue that the ontology to be chosen should be able to describe the time an event or a measurement took place instead of relying to timestamps that usually denote the time of arrival of a stream element. The use of such timestamps does not help sort out the true sequence of events since the respective notifications may arrive in disorder due to variable delays in the routing from the source to the receiver. Ensuring that incoming information is already annotated with its time of occurrence handles elegantly the infamous event disorder issue in data streams, under the assumption that all disparate sources share a common time reference frame.

3. *A semantic ruleset.* Generally, the inclusion of a set of rules encoding the application or domain-specific knowledge is optional and depends on the application designer's choice. In the SensorStream case, semantic rules are used to recognize events or situations of interest and perform some action according to the current application (e.g. issue of an alert). The form and syntax of these rules resembles to the one of the mapping rules presented above, with the difference that, in the case of the former, the condition refers to the ontology model. Example conditions include checks on the existence or number of individuals for a given class or checks on the existence of a non-empty result set for a given SPARQL query. Therefore, the *condition* part of SensorStream's semantic rules can be replaced by an appropriate SPARQL/Update query, guaranteeing that an expressiveness equal to that of SPARQL is achieved.

Windowing is an important decision that has to be taken since, by definition, streams are unbounded and cannot fit in memory in order to be processed as usual. Windowing concerns three important decisions that have to be taken:

1. *The window measurement unit.* This choice refers to the basic unit serving as the window definition basis. In this aspect, a window can be either physical or logical. Physical windows are defined in terms of physical units, which may be tuples in the case of DSMS or RDF triples in the Semantic Web context. Logical windows, on the other hand, as already stated in Section 2, are typically defined in terms of time units.

SensorStream follows a different path, by defining windows in terms of the number of knowledge base individuals they refer to. This point of view is significantly different from triple-based physical windows and offers significant benefits. An individual-based window may not have a fixed statement length but instead guarantees that all statements involving an individual are kept in the current window range. On the contrary, in the triple-based physical window case, statements are dropped out of the window based solely on their timestamp and not on their relationship with other statements. For example, an incoming statement might refer to an individual that participates in older statements already dropped out of the triple-based window and hence, knowledge about it is incomplete. An individual-based window decreases considerably the degree of incompleteness by dropping out groups of statements that refer to the oldest individual in the knowledge base.

An *individual-based window* is defined over a stream of RDF triples, where every element of the stream can be written as $(< s, p, o >, \tau)$. We first introduce the *individual timestamp* t_{ind} for a given entity s and time τ as the timestamp of its most recent appearance as subject in some RDF triple of stream $\mathbb{S}$ measured in timestamp τ:

$$t_{ind}(s,\tau) = \{max(t) : (< s, p, o >, t) \in \mathbb{S} \wedge t \leq \tau\} \quad (6)$$

We then define an individual-based logical window as the set of the RDF triples that contain as subject an entity with individual timestamp t_{ind} occurring in the $[t_1, t_2]$ interval:

$$w'_l(\mathbb{S}, t_1, t_2) = \{(< s, p, o >, \tau) \in \mathbb{S} : t_1 \leq t_{ind}(s, t_2) \leq t_2\} \quad (7)$$

From the above, similarly to Equation (2), we can define an individual-based physical window of size N as the set of the RDF triples that contain as subject one of the N most recently appeared individuals in stream $\mathbb{S}$:

$$w'_p(\mathbb{S}, N, \tau) = \{w'_l(\mathbb{S}, t_1, \tau) : t_1 \leq \tau \wedge |w'_l(\mathbb{S}, t_1, \tau)|_I \leq N \wedge \forall t_2 < t_1 : |w'_l(\mathbb{S}, t_2, \tau)|_I > N\} \quad (8)$$

where $|\cdot|_I$ denotes the size in terms of individuals. The definitions for the sliding, tumbling and landmark logical windows given in Section 2 can be adapted to the individual-based case, by replacing typical tuple-based logical windows w_l with individual-based ones w'_l and expressing window size w and progression step d as a number of individuals.

2. *The window size.* We need to note that the window size causes a proportional increase in the system latency and therefore these two parameters have to be balanced accordingly. On one hand, in order to process data in real time we would like to have all the information available in memory but on the other hand latency should be restricted to acceptable levels.

3. *The window behavior.* In (Konstantinou et al., 2010) a system is investigated where the buffer is "flushed" after the processing latency exceeded certain thresholds. This behavior, although convenient for a large number of cases, it is not always optimal since maintenance operations could take place at inconvenient times (e.g. during the occurrence of an event of interest) leading to incorrect results. The approach followed in SensorStream deals with this problem by maintaining the latest information at all times.

As we will see in the next section, the application of a window ensures that the response time stays within some finite bounds. This, of course, happens at the expense of the amount and completeness of information stored in the cache Knowledge Base (KB), upon which the semantic rules are applied. This clearly prevents us from declaring rules that make use of older information and thus, we rely solely on the information kept inside the window. Depending on the application requirements, different choices of the window size and the amount of information available at real-time can be made. For example, in a person tracking application taking place in a crowded area, a window of 100 ontology individuals might prove insufficient for real-time recognition of high-level events, while the same window would probably be satisfactory for the real-time monitoring of a natural phenomenon. This prevents us from establishing a general application quality metric, although we could risk saying that in most cases, application quality is positively correlated with the completeness of information and, therefore, with window size as well.

Regardless of the window choice, and in contrast with most systems employing windowing techniques, SensorStream stores older elements that are dropped out of the current window in a persistent Knowledge Base to enable historical offline queries, as shown in Figure 2.

In brief, the course of actions taken by our SensorStream prototype as a new information element arrives is illustrated in Algorithm 1. When a new stream element becomes available, reasoning on the current instance of the cache Knowledge Base must be performed to ensure that all inferences regarding the new element are computed. Fortunately, the redundancy implied in reasoning over successive Knowledge Base instances can be handled by a reasoner (e.g. Pellet) employing incremental reasoning techniques. This step is essential in order to build a system that is both knowledge-aware and real-time, given that:

- A simple unprocessed message insertion in the cache knowledge base does not render the system knowledge-aware, since no implicit information can be extracted and added as well, without use of a reasoner.

- The notion of real-time is tightly coupled with the concepts of *event* and *response time*. More precisely, according to (Dougherty and Laplante, 1995), an *event* can be defined as "any occurrence that results in a change in the sequential flow of program execution" and the *response time* as "the time between the presentation of a set of inputs and the appearance of all the associated outputs". Therefore, without reloading the entire Knowledge Base and performing reasoning on it, we cannot consider that all the associated outputs are produced and the system cannot be classified as real-time. Systems that perform simple insertions upon each message arrival and scheduled inferencing at predefined time intervals cannot be considered real-time, only "near real-time" if these intervals are frequent enough.

After the reasoning procedure is over, the current snapshot of the window has to be updated to include the new element and possibly drop older elements, depending on the type of window chosen. In the same time, the elements that were dropped out of the window are moved to the persistent Knowledge Base for future reference. Then, the mapping rules are applied to populate the system's cache Knowledge Base. Reasoning and semantic rules' check is performed afterwards in order to spot high-level events. As we have already mentioned, semantic rules can be replaced by an equivalent SPARQL query. Since semantic rules are constantly checked for every incoming element, they in fact encapsulate continuous queries running against the temporary cache knowledge base.

Algorithm 1 Process Element

```
if new element arrives then
  load cache Knowledge Base
  compute new window
  copy outdated information to the persistent Knowledge Base
  apply mapping rules
  apply semantic rules
end if
```

5 Performance Evaluation

5.1 Measurement environment

For our measurements we assumed a simple object tracking application scenario. A data stream in the form of XML messages containing sensor data is received from

the server for real time semantic processing and event recognition. Continuous updates in the ontology carried out by the execution of the mapping rules allow the knowledge base to evolve using the information from the data stream while the semantic rules provide a means for the use of the complete (initial and inferred) knowledge for rule-based inferencing, continuous query answering and event recognition.

The data stream is simulated by means of a client application who produces and sends at predefined time intervals XML messages with dummy content that conform to a specific template shown in the next subsection.

Our measurements were taken in a lab environment. The server was an Intel® Core™ 2 Quad@2.83GHz, with 3962 MB total memory running a x86_64 Ubuntu 9.10 OS. The client was running on an Intel® Core™ 2 Duo@2.00GHz, with 2012 MB total memory and 32 bit Ubuntu 9.10 OS. The core components of the Annotation and Knowledge layers of the SensorStream system, were implemented using Jena 2.6.2, ARQ 2.8.1 and Pellet 2.0.2 libraries, and MySQL 5.1.37 server.

5.2 Measurement methodology

First, we configure the system with a set of initial, necessary inputs, that is the ontology, the mapping rules and the semantic rules. The ontology used was inspired from the context awareness domain, including concepts like `Actor`, `Location`, and relevant properties like `hasTime`, `hasLocation`, etc. The ontology was designed to be generic enough to support a system for human or object tracking in general. Since the development of a more expressive and richer ontology is out of the scope of this work, it was kept as minimal as possible, using only the lightweight $\mathcal{ALC}$(D) Description Logic language, without General Concept Inclusions (GCIs) and numbering 31 classes, 3 object properties and 7 datatype properties, in a total of 110 statements.

The message template of the incoming XML stream, as shown below with a message example, includes information about the location, the time and the confidence of the measurement.

```
<?xml version='1.0' encoding='UTF-8'?>
<Message>
   <Event id='9'>
      <Tracker type='FaceTracker'>
         <TimeStamp value = '2678'/>
         <person id='2678' certainty='100' >
            <location2d x='72' y='8' />
         </person>
      </Tracker>
   </Event>
</Message>
```

The XML stream used as the input set for the purpose of our measurements contains 4000 messages having the same structure as the above. The mapping rules for this template and ontology, provide the assertion of an individual of type `Human`, with the extracted values from the XML message serving as values for the datatype properties `hasTime`, `hasXLocation`, `hasYLocation`, `hasCertainty` and `hasName`. The semantic rule in our measurements scenario is just checking for the existence of individuals of the Human class and if there is one or more, it adds exactly one in the `Action` class. As the performance of the system highly depends on the number and the complexity of rules, the rationale behind the experiments run was, on one hand, to test the performance of the system with the presence of all the features of our architecture but on the other hand, fully control the complexity introduced by the rules. Thus, our mapping rule inserts 5 statements with every incoming message: one for the type definition and four for the assignment of properties' values, while our semantic rule leaves the knowledge base intact, with the exception of the first run when the class of interest contains no individuals.

For each experiment, the main variable of interest is latency. Informally, we define latency as the time elapsed between the transmission of an XML message from the information source and the execution of the last semantic rule in this processing cycle. This time period t_{total} consists of processing time t_p performed by the system and, to a smaller extent, message transmission delay t_{tr} incurred by the network, as shown in Equation (9).

$$t_{total}(N) = t_{tr} + t_p(N) \tag{9}$$

$$t_p(N) = t_i(N) + t_w(N) + t_m(N) + t_s(N) \tag{10}$$

Furthermore, t_p consists of the initialization interval t_i, during which the cache KB is loaded into the memory, the window selection interval t_w, when the time bounds of the window and its contents are updated and, finally, the mapping and semantic rules execution intervals, t_m and t_s respectively. All the time variables introduced in Equation (10) are functions of the window size N (in terms of statements). For every experiment run, we measure the total latency and the process time with respect to the number of incoming statements, as shown in the next subsection. We should clarify here that we restrict ourselves to one scenario, given that our intention is the comparison of the overall system performance with respect to different window types and varying window sizes. A thorough benchmarking of the system performance for a given window type would involve the investigation of different scenarios with several different application cases of mapping and semantic rules, varying ontology size and number of statements to number of individuals ratio, and is thus considered out of scope for the current paper.

5.3 Results

First, we perform a test in order to assess the system's behaviour in the absence of a window. In Figure 3 we illustrate the system's behaviour with Pellet, that

employs incremental reasoning techniques to reduce the total time of the reasoning process. Although this feature of Pellet reduces radically the response time of the server and its performance is superior to static tableau-based reasoning algorithms, it still does not suffice for meeting the real-time requirements in the case of streaming data since processing time is shown to increase steadily with the number of statements in the knowledge base. As expected, the difference between total latency and processing time, which accounts for the network delay, is small and constant throughout the entire test.

For this purpose, we employ and study the efficiency of two windowing approaches, where the size of the windows used are measured in terms of the number of individuals stored in the cache KB, as mentioned in Section 4. We try an *individual-based sliding window* with unit advancement step and an *individual-based "gradually filling" tumbling window* and examine how their size affects the process time of the system. The advancement step in the sliding window is also defined in terms of individuals, in other words as a new individual becomes available, the sliding window progresses by excluding statements involving the oldest individual in the knowledge base and including statements referring to the new one. A random selection occurs when there are more than one individuals with the oldest timestamp. Attention though must be drawn to the fact that the process of maintaining and constantly updating a window of the incoming stream in-memory places an additional burden for the knowledge base and constitutes an overhead for the system.

The *"gradually filling" tumbling window* has, like all tumbling windows, a progression step equal to the window size, but fills gradually as individuals arrive to the knowledge base instead of waiting to be completely filled at once. For ease of representation, we give the definition of a logical "gradually filling" tumbling window, which can be easily adjusted for the individual-based case:

$$w_{tum,gf}(\mathbb{S}, w, t_0, \tau) = \begin{cases} w_l(\mathbb{S}, t_0, \tau), & t_0 \leq \tau < t_0 + w \\ w_l(\mathbb{S}, \tau - mod(\tau - t_0, w), \tau), & \\ \qquad t_0 + w \leq \tau \text{ and } mod(\tau - t_0, w) \neq 0 & \\ w_l(\mathbb{S}, \tau - w, \tau), t_0 + w \leq \tau \text{ and } mod(\tau - t_0, w) = 0 & \end{cases} \tag{11}$$

While formulating a window over a stream in DSMS is a straightforward procedure, the same does not apply to the case of a knowledge base. The initial formulation and constant update of the window is performed with the aid of SPARQL queries that are kept as lightweight as possible in order to minimize the overhead imposed by their execution. For our measurements, the sliding window was implemented indirectly using the following SPARQL query to select the older individual in the knowledge base. All the statements involving the selected individual are inserted in the persistent KB and then removed from the cache KB. This SPARQL query is executed every time a new individual appears, given the unit progression step of the sliding window. As mentioned in Section 4, the `hasTime` property present in our testing ontology is the property that time-annotates an individual. In our case, this annotation refers to the generation time of the incoming message.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?x
  WHERE {?x :hasTime ?time}
ORDER BY xsd:integer(?time) LIMIT 1
```

Equivalently, we could have used the following SPARQL/Update query to perform in one step the same procedure[4]. However, this query, more complex as it is with the use of the `OPTIONAL` pattern matching (Perez et al., 2006), takes considerably more time to execute on Jena's ARQ engine than the previous one. Therefore, we decided to stick to the indirect two-step windowing procedure.

```
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

INSERT
  { GRAPH <http://sensorstream/persistent>
   {?x ?y ?z} }
WHERE
  { GRAPH  <http://sensorstream/cache>
   {?x :hasTime ?time .
    ?x ?y ?z .
   OPTIONAL { ?older :hasTime ?otime .
   FILTER (xsd:integer(?otime) <
    xsd:integer(?time))}
FILTER (!bound(?older))}

WITH <http://sensorstream/cache>
DELETE
   {?x ?y ?z}
WHERE
   {?x :hasTime ?time .
    ?x ?y ?z .
   OPTIONAL { ?older :hasTime ?otime .
   FILTER (xsd:integer(?otime) <
     xsd:integer(?time))}
FILTER (!bound(?older))}
```

In a similar fashion, for the maintenance of the "gradually filling" tumbling window, we used the following SPARQL query to first select all individuals from the cache KB, copy them to the persistent KB and finally, remove them from the cache KB.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syn
tax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ?x
WHERE
  {?class rdf:type owl:Class. ?x rdf:type ?class}
```

In the following set of experiments, we show in Figures 4 and 5 the performance of the system with the

use of tumbling and sliding windows for sizes of 1000, 5000 and 15000 triples or 191, 991 and 2991 individuals, respectively. Due to the specific mapping rules used in the experiment, as explained in Section 5.2, every individual is involved in 5 statements, hence the number of statements is approximately 5 times the number of individuals considered. Next to each performance diagram, we append the evolution of the number of statements with the number of incoming messages to clearly show the performance dependency on the number of statements in the cache KB.

The peaks that are seen in the process time in Figure 4 show that in the case of a tumbling window, when reaching the window size and entering a new window computation stage, the time needed to clear the window contents is much higher than the typical time needed to process a single message. This delay is caused by the fact that in this maintenance stage every individual in the cache KB is deleted and the whole model of the ontology is affected. Moreover, as we can see in Figures 4a, 4c, 4e and Table 1, the processing time at the maintenance stage increases at a high and growing rate, meaning that a tumbling window needs further improvements in order to scale for larger datasets. Apart from the performance, the choice of a tumbling window entails the possibility of producing a false alarm since, after a maintenance, the ontology will be back to a pristine state and all previous information that could possibly lead to an alarm will no longer be available. Therefore, the use of such a tumbling window is recommended in cases where a small loss in captured events is not crucial.

Table 1 Processing time for tumbling and sliding windows of varying size (measured in number of statements). Average gradients are shown in parentheses

Window Size	t_p^{tum}	t_p^{sl}
1000	1250 ms (1.005)	253 ms (0.135)
5000	5270 ms (1.108)	793 ms (0.153)
15000	16348 ms	2306 ms

On the other hand, the behaviour of a sliding window is more "predictable" and closer to the desired one, compared to the tumbling window. In Figures 5a, 5c, 5e, the system shows a stable performance, where the overhead of computing the new window instance is kept low regardless the size of the window. The average process time, t_p^{sl}, in the case of the sliding window, as shown in Table 1, increases at a relatively small and almost stable rate as a function of the window size. While keeping a stable response, the system is able to make real-time inferences from the newly asserted data, providing not only improved performance, but also more accurate real-time entailments. The peaks that are seen in the figures are caused by turbulences in the testing environment, leading to variations in network delay.

Finally, in Figure 6, the performance of the two windowing approaches are juxtaposed to the plain no window approach for the three window sizes considered to show the superior performance of the sliding window approach. The above results, strengthen the view that a windowing approach over a KB, can be a stable solution to Semantic Web applications where time is a critical parameter. Furthermore, our individual-based window approach provides hints that Semantic Web techniques can be applied in order to exploit stream semantics and ameliorate application level quality given the latency limitation posed by certain real-time applications.

6 Conclusions and Future Work

As streams of data become increasingly available, real-time solutions are needed to process and combine streams in order to extract high-level events with minimal human intervention. In this paper, we presented a semantic-based framework for dealing with information streams from heterogeneous sources encoded in various XML templates and its prototype implementation, SensorStream. Our framework combines consistently a number of technologies leading to a powerful solution for the problem of combining semantic information in real time. SensorStream does not restrict itself to an application domain, as the ontology encoding the knowledge of the domain is configurable and can be selected and tailored to the application needs. Moreover, it allows for multiple information sources that store their output in possibly distinct XML templates. The syntactic heterogeneity of the latter is handled via mapping rules that populate the domain ontology with appropriate individuals. We argued that, given a common time reference frame shared by all information sources, incorporating the notion of time in the ontology allows for annotating each event with its time of occurrence. This design choice differs significantly from the traditional timestamps of stream management systems that are able to capture only the time of arrival of every message to the system, giving rise to the event disorder problem.

Two windowing techniques that restrict the unbound stream to a finite stream subset were borrowed from the related stream management research field in databases and adjusted accordingly to the knowledge base context. What needs to be noted though is the fact that we do not drop the elements that exit the window but we store them explicitly in a persistent knowledge base to allow for future reuse, reference and execution of historical queries on them. Tests were performed for various sizes of these two window types, namely the sliding and "gradually filling" tumbling window, and the performance of the system was measured. Results have shown that the sliding window performs better than the tumbling window and thus, its application meets the real-time requirements for semantic processing of information streams. Average process time increases, as anticipated, as the applied window grows larger, but, as

the tests suggest, at a lower rate than the window size increase rate.

Next steps extending SensorStream's functionality would include testing of other window types mentioned in the stream database literature, such as landmark windows, while we could try to make the appropriate adjustments to other related techniques that reduce the volume of a stream, such as filtering or synopses. It would be interesting to explore how these techniques would be applied in a knowledge base, taking advantage of the elements' properties and the relationships linking them in order to achieve smarter load shedding. Furthermore, more extensive tests of our framework could be performed examining how the ontology's complexity and size as well as the complexity of the applied rules affect the overall system's performance.

Acknowledgements

Dimitrios-Emmanuel Spanos wishes to acknowledge the financial support of "Alexander S. Onassis Public Benefit Foundation", under its Scholarships Programme.

References

Abadi, D., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S. (2005). 'The Design of the Borealis Stream Processing Engine.' In: *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA*. ACM.

Allen, B. D., Bishop, G. and Welch, G. (2001). 'Tracking: Beyond 15 minutes of thought.' *SIGGRAPH Course Pack*.

Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U. and Widom, J. (2004). 'STREAM: The Stanford Data Stream Management System.' Technical report, Stanford InfoLab.

Arasu, A., Babu, S. and Widom, J. (2006). 'The CQL Continuous Query Language: Semantic Foundations and Query Execution.' *The VLDB Journal*, vol. 15, no. 2, pp. 121–142.

Baader, F., Horrocks, I. and Sattler, U. (2005). *Description Logics as Ontology Languages for the Semantic Web*. Heidelberg: Springer Berlin, pp. 228–248.

Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J. (2002). 'Models and Issues in Data Stream Systems.' *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '02*, pp. 1–16.

Barbieri, D., Braga, D., Ceri, S., Della Valle, E. and Grossniklaus, M. (2009). 'C-SPARQL: SPARQL for Continuous Querying.' In: *Proceedings of the 18th International Conference on World Wide Web*. ACM New York, NY, USA, pp. 1061–1062.

Barbieri, D., Braga, D., Ceri, S. and Grossniklaus, M. (2010). 'An Execution Environment for C-SPARQL Queries.' In: *Proceedings of the 13th International Conference on Extending Database Technology*. ACM, pp. 441–452.

Bizer, C., Heath, T. and Berners-Lee, T. (2009). 'Linked Data – the Story so Far.' *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22.

Bolles, A., Grawunder, M. and Jacobi, J. (2008). 'Streaming SPARQL - Extending SPARQL to Process Data Streams.' In: *The Semantic Web: Research and Applications*. Springer, pp. 448–462.

Bradski, G. R. (1998). 'Computer Vision Face Tracking for Use in a Perceptual User Interface.' Intel Technology Journal.

Broekstra, J., Kampman, A. and van Harmelen, F. (2002). 'Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema.' In: Horrocks, I. and Hendler, J. (eds.), *Proceedings of the first Int'l Semantic Web Conference (ISWC 2002)*, vol. 2342 of *Lecture Notes in Computer Science*. Sardinia, Italy: Springer Verlag, pp. 54–68.

Calder, M., Morris, R. A. and Peri, F. (2010). 'Machine Reasoning about Anomalous Sensor Data.' *Ecological Informatics*, vol. 5, no. 1, pp. 9–18.

Carroll, J., Dickinson, I., Dollin, C. and Reynolds, D. (2004). 'Jena: Implementing the Semantic Web Recommendations.' In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. pp. 74–83.

Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M., Hellerstein, J., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F. and Shah, M. (2003). 'TelegraphCQ: Continuous Dataflow Processing for an Uncertain World.' In: *Proceedings of the Conference on Innovative Data Systems Research (CIDR 2003)*. ACM Press.

Chen, J., DeWitt, D., Tian, F. and Wang, Y. (2000). 'NiagaraCQ: A Scalable Continuous Query System for Internet Databases.' *ACM SIGMOD Record*, vol. 29, no. 2, pp. 379–390.

Comaniciu, D. and Meer, P. (1999). 'Mean shift analysis and applications.' In: *Proceedings of the Seventh IEEE International Conference on Computer Vision (ICCV'99)*, vol. 2. Kerkyra, Greece: IEEE Computer Society, pp. 1197–1203.

Conolly, D. (2007). 'Gleaning Resource Descriptions from Dialects of Languages (GRDDL).' Available online at `http://www.w3.org/TR/grddl/`.

Cuenca Grau, B., Halaschek-Wiener, C. and Kazakov, Y. (2007). 'History Matters: Incremental Ontology Reasoning Using Modules.' In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G. and Cudr-Mauroux, P. (eds.), *Proceedings of the 6th International Semantic Web Conference (ISWC 2007)*, vol. 4825 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pp. 183–196.

Das, S. and Srinivasan, J. (2009). 'Database Technologies for RDF.' In: *5th International Summer School 2009 on Reasoning Web. Semantic Technologies for Information Systems, LNCS 5689*. pp. 205–221.

Della Valle, E., Ceri, S., van Harmelen, F. and Fensel, D. (2009). 'It's a Streaming World! Reasoning upon Rapidly Changing Information.' *Intelligent Systems, IEEE*, vol. 24, no. 6, pp. 83–89.

Demers, A., Gehrke, J., Panda, B., Riedewald, M. and Sharma, V. (2007). 'Cayuga: A General Purpose Event Monitoring System.' In: *Proc. CIDR*. New York, New York, USA: ACM Press, pp. 412–422.

Dougherty, E. and Laplante, P. (1995). *Introduction to Real-Time Imaging*, chap. What is Real-Time Processing? Wiley-IEEE Press, pp. 1–9.

Erling, O. and Mikhailov, I. (2010). *Virtuoso: RDF Support in a Native RDBMS*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 501–519.

Ghanem, T., Aref, W. and Elmagarmid, A. (2006). 'Exploiting Predicate-Window Semantics over Data Streams.' *ACM SIGMOD Record*, vol. 35, no. 1, pp. 3–8.

Gutierrez, C., Hurtado, C. and Vaisman, A. (2007). 'Introducing Time into RDF.' *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 2, pp. 207–218.

Gyllstrom, D., Wu, E., Chae, H., Diao, Y., Stahlberg, P. and Anderson, G. (2007). 'SASE: Complex Event Processing over Streams.' In: *Proceedings of CIDR*. ACM Press.

Halaschek-Wiener, C., Parsia, B. and Sirin, E. (2006). 'Description Logic Reasoning with Syntactic Updates.' In: Meersman, R. and Tari, Z. (eds.), *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, vol. 4275 of *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 722–737. URL `http://www.springerlink.com/content/g4p2031727154281`.

Heintz, F., Kvarnstrom, J. and Doherty, P. (2009). 'Stream Reasoning in DyKnow: A Knowledge Processing Middleware System.' In: *1st International Workshop on Stream Reasoning*. Springer.

Henson, C., Neuhaus, H., Sheth, A., Thirunarayan, K. and Buyya, R. (2009). 'An Ontological Representation of Time Series Observations on the Semantic Sensor Web.' In: *Proceedings of the 1st Int. Workshop on the Semantic Sensor Web (SemSensWeb), collocated with ESWC*.

Hobbs, J. and Pan, F. (2006). 'Time Ontology in OWL.' Available online at `http://www.w3.org/TR/owl-time/`.

Kokar, M., Matheus, C. and Baclawski, K. (2009). 'Ontology-based Situation Awareness.' *Information fusion*, vol. 10, no. 1, pp. 83–98.

Konstantinou, N., Solidakis, E., Zafeiropoulos, A., Stathopoulos, P. and Mitrou, N. (2010). 'A Context-Aware Middleware for Real-Time Semantic Enrichment of Distributed Multimedia Metadata.' *Multimedia Tools and Applications*, vol. 46, no. 2-3, pp. 425–461.

Le-Phuoc, D. and Hauswirth, M. (2009). 'Linked Open Data in Sensor Data Mashups.' In: *Proceedings Semantic Sensor Networks*.

Li, J., Maier, D., Tufte, K., Papadimos, V. and Tucker, P. A. (2005). 'Semantics and Evaluation Techniques for Window Aggregates in Data Streams.' *Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05*, pp. 311–322.

Lien, C. C., Chiang, C. L. and Lee, C. H. (2007). 'Scene-based Event Detection for Baseball Videos.' *Journal of Visual Communication and Image Representation*, vol. 18, no. 1, pp. 1–14.

Liggins, M. E., Hall, D. L. and Llinas, J. (eds.) (2009). *Handbook of Multisensor Data Fusion: Theory and Practice*. CRC Press.

Little, E. and Rogova, G. (2009). 'Designing Ontologies for Higher Level Fusion.' *Information Fusion*, vol. 10, no. 1, pp. 70–82.

Liu, L., Pu, C. and Tang, W. (1999). 'Continual Queries for Internet Scale Event-Driven Information Delivery.' *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 4, pp. 610–628.

Maier, D., Tucker, P. and Garofalakis, M. (2005). 'Filtering, Punctuation, Windows and Synopses.' In: *Stream Data Management*. pp. 35–58.

Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A. and Lutz, C. (2009). 'OWL 2 Web Ontology Language Profiles.' Available online at `http://www.w3.org/TR/2009/PR-owl2-profiles-20090922/`.

Motik, B., Sattler, U. and Studer, R. (2005). 'Query Answering for OWL-DL with rules.' *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 1, pp. 41–60.

Papamarkos, G., Poulovassilis, A. and Wood, P. T. (2003). 'Event-Condition-Action Rule Languages for the Semantic Web.' In: *Workshop on Semantic Web and Databases*. Springer, pp. 309–327.

Parsia, B., Halaschek-Wiener, C. and Sirin, E. (2006). 'Towards Incremental Reasoning Through Updates in OWL-DL.' In: *Proceedings of the Reasoning on the Web Workshop at WWW 2006*. pp. 1–6.

Patroumpas, K. and Sellis, T. (2006). 'Window Specification over Data Streams.' In: *International Conference on Semantics of a Networked World: Semantics of Sequence and Time Dependent Data (ICSNW'06)*. Springer, pp. 445–464.

Perez, J., Arenas, M. and Gutierrez, C. (2006). 'Semantics and Complexity of SPARQL.' In: *Proceedings of the International Semantic Web Conference (ISWC 2006)*. Springer, pp. 30–43.

Prud'hommeaux, E. and Seaborne, A. (2008). 'SPARQL Query Language for RDF.' Available online at `http://www.w3.org/TR/rdf-sparql-query/`.

Pugliese, A., Udrea, O. and Subrahmanian, V. S. (2008). 'Scaling RDF with Time.' *Proceedings of the 17th international conference on World Wide Web - WWW '08*, pp. 605–614.

Rodriguez, A., McGrath, R., Liu, Y. and Myers, J. (2009). 'Semantic Management of Streaming Data.' In: *Workshop on Semantic Sensor Nets at International Semantic Web Conference*. CEUR-WS.org.

Sirin, E., Parsia, B., Kalyanpur, A., Grau, B. and Katz, Y. (2007). 'Pellet: A Practical OWL-DL Reasoner.' *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53.

Tsarkov, D. and Horrocks, I. (2006). 'FaCT++ Description Logic Reasoner: System Description.' In: *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006), volume 4130 of Lecture Notes in Artificial Intelligence*. Springer, pp. 292–297.

Volz, R., Staab, S. and Motik, B. (2005). 'Incrementally maintaining materializations of ontologies stored in logic databases.' *Journal on Data Semantics II*, pp. 1–34.

Walavalkar, O., Joshi, A., Finin, T. and Yesha, Y. (2008). 'Streaming Knowledge Bases.' In: *Proceedings of the Fourth International Workshop on Scalable Semantic Web knowledge Base Systems*. Springer.

Whitehouse, K., Zhao, F. and Liu, J. (2006). 'Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data.' In: *Lecture Notes In Computer Science 3868*. Springer, pp. 5–20.

Zhang, D. and Chang, S. F. (2002). 'Event Detection in Baseball Video Using Superimposed Caption Recognition.' In: *Proceedings of the tenth ACM international conference on Multimedia*. ACM New York, NY, USA, pp. 315–318.

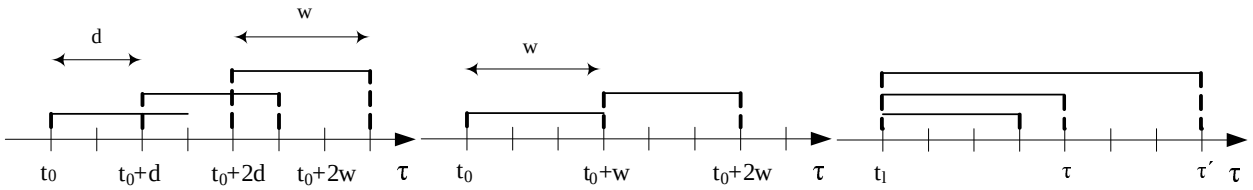
Notes

[1] W3C Web Services Activity, `http://www.w3.org/2002/ws/`

[2] NoSQL databases do not typically provide any declarative query language equivalent to SQL.

[3] OWLJessKB, `http://edge.cs.drexel.edu/assemblies/software/owljesskb/`

[4] Since Jena's ARQ engine does not allow update nested queries, the alternative is a much more unintuitive query which finds as well the oldest individual in the knowledge base



(a) Sliding window with size w and progression step d (b) Tumbling window with size w (c) Landmark window with lower bound t_l

Figure 1: Time-based window types

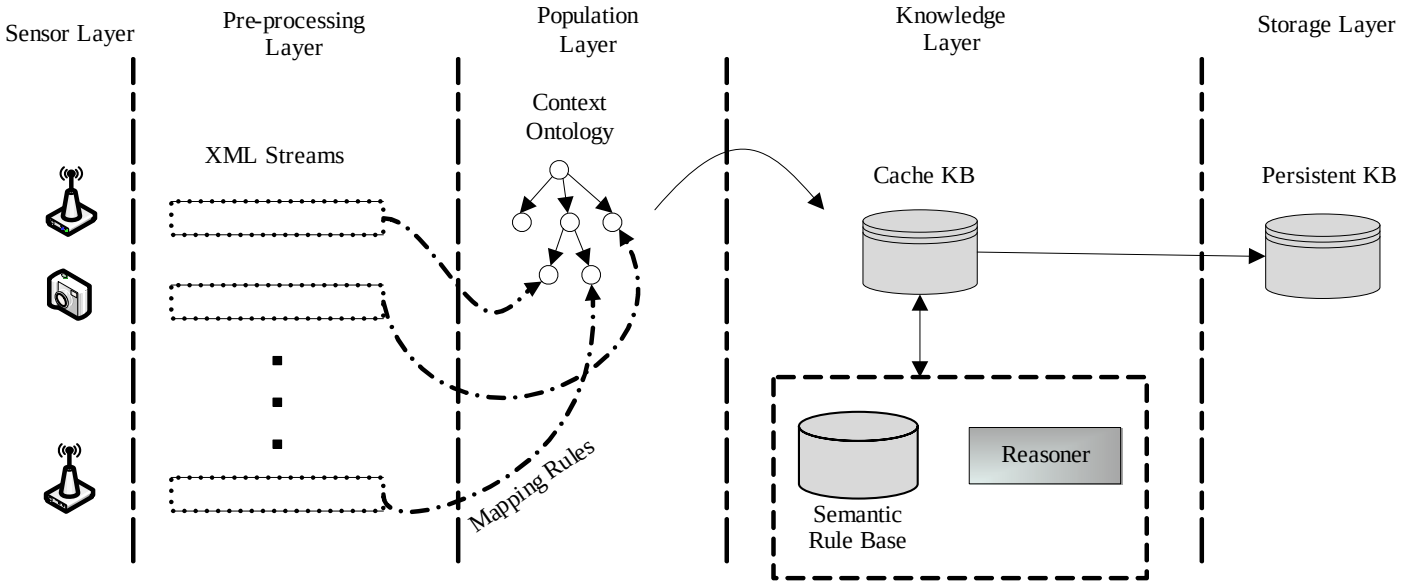
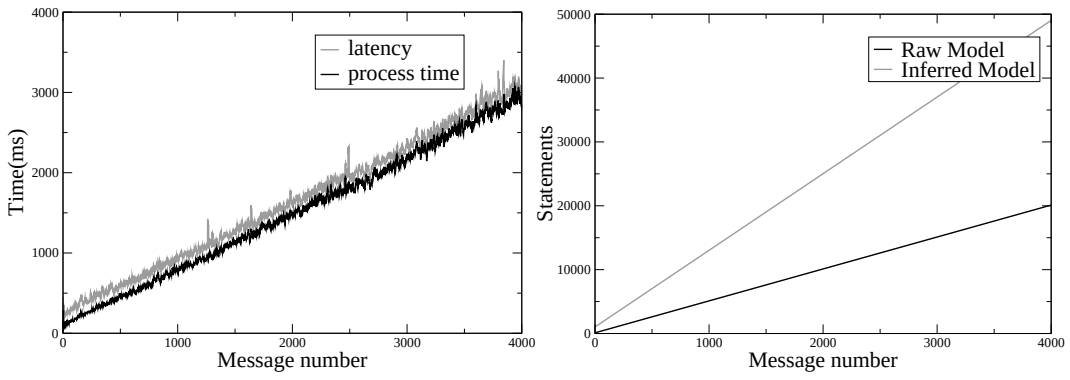
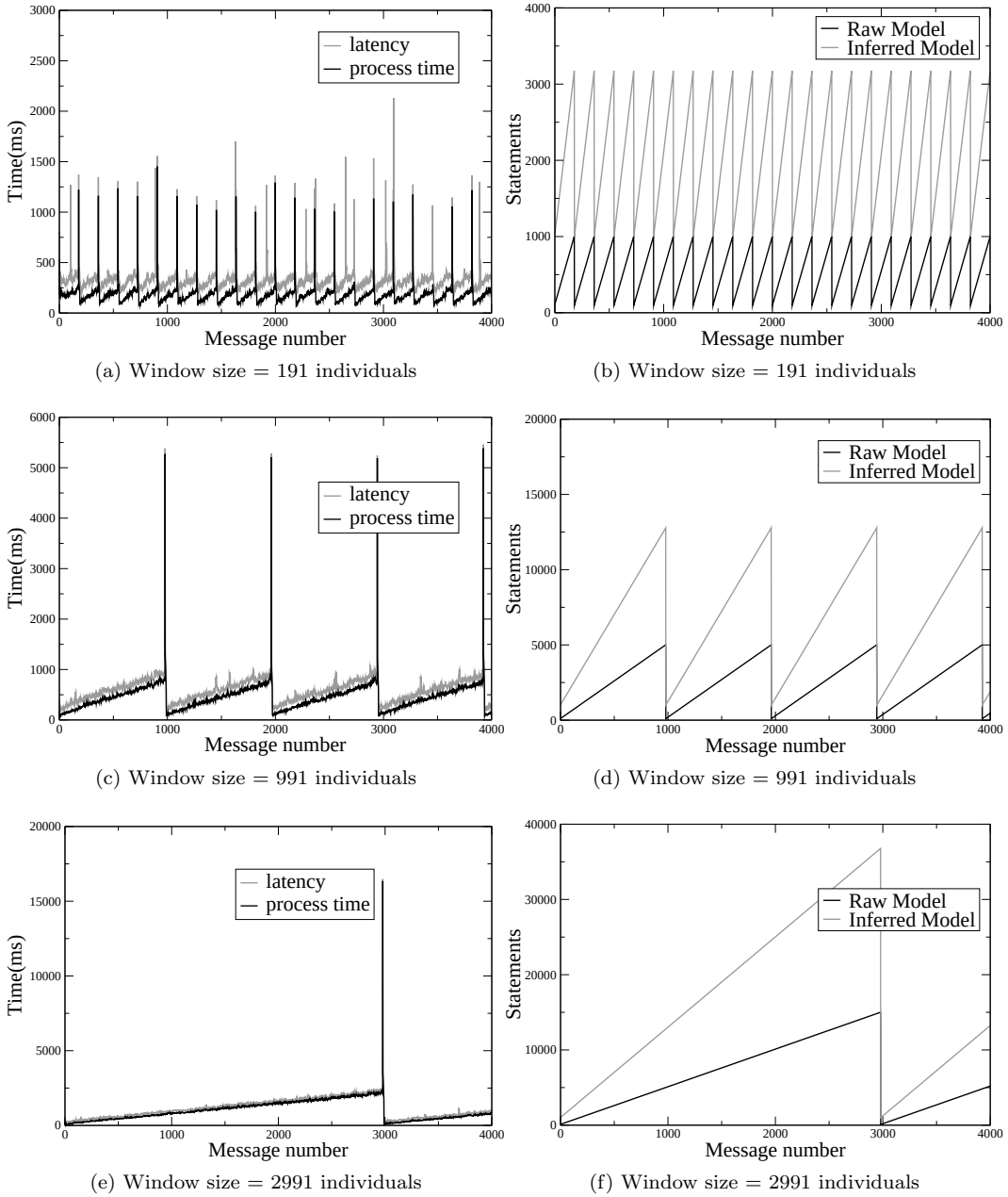


Figure 2: Information flow in SensorStream



(a) Process time and latency (b) Number of statements

Figure 3: Performance and number of statements without windowing. Raw model refers to the initial model while inferred refers to the model after the reasoning procedure.



(a) Window size = 191 individuals

(b) Window size = 191 individuals

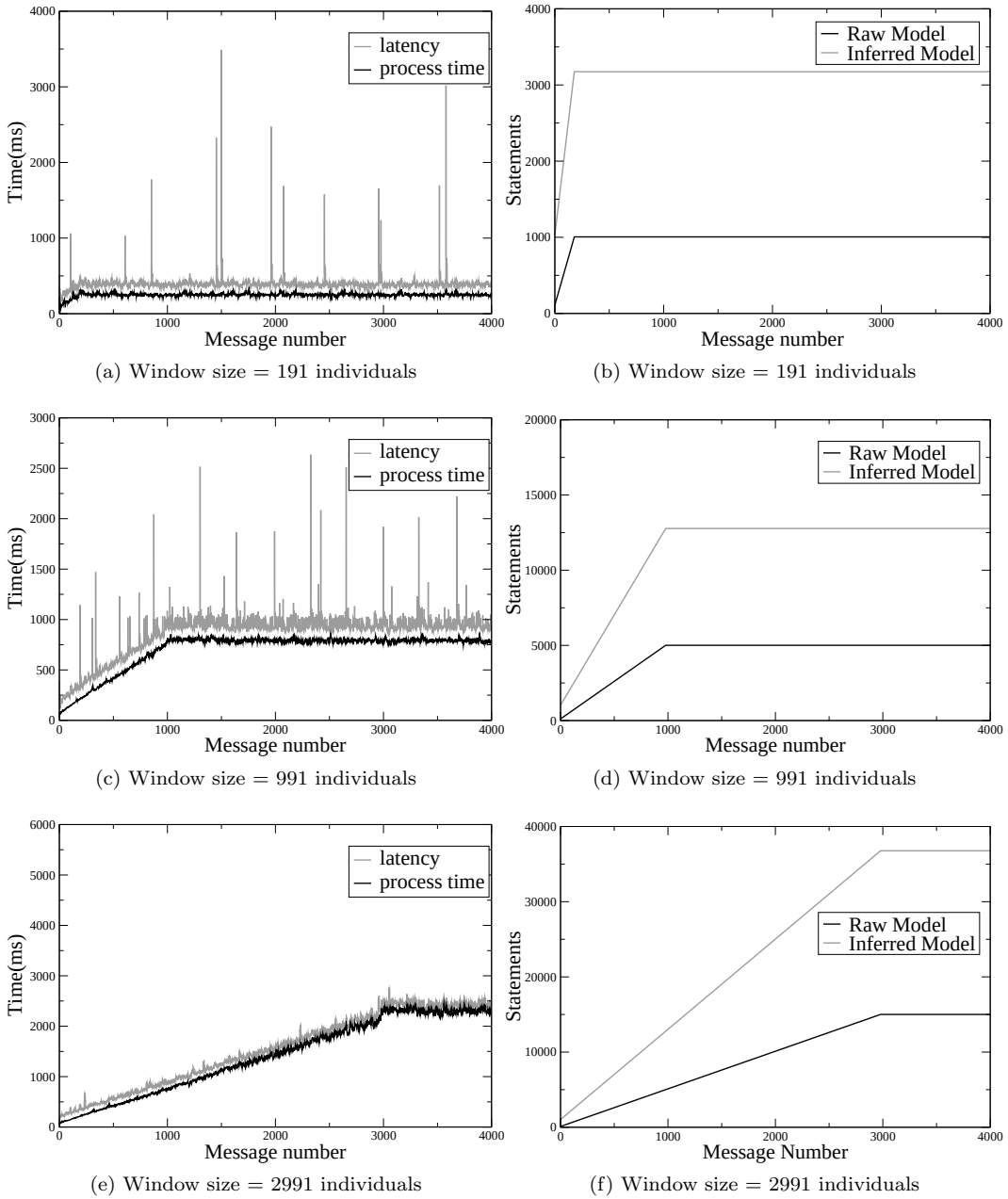
(c) Window size = 991 individuals

(d) Window size = 991 individuals

(e) Window size = 2991 individuals

(f) Window size = 2991 individuals

Figure 4: Performance and number of statements for "gradually filling" tumbling window of various sizes. Raw model refers to the initial model while inferred refers to the model after the reasoning procedure.



(a) Window size = 191 individuals

(b) Window size = 191 individuals

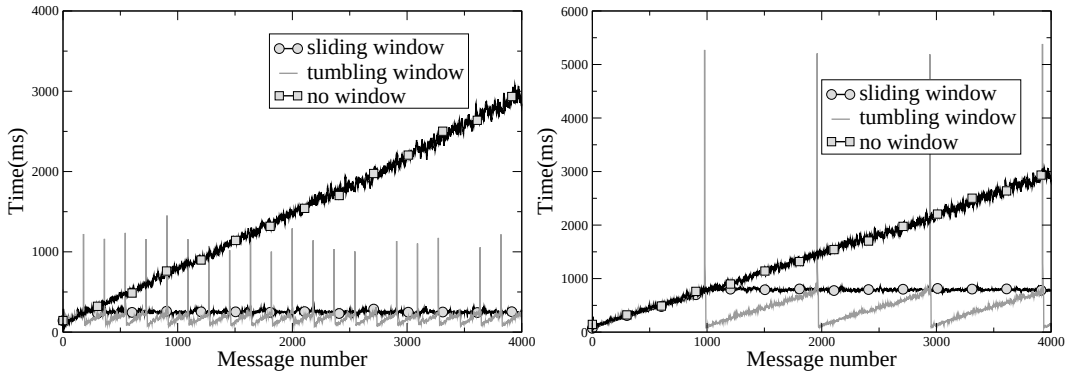
(c) Window size = 991 individuals

(d) Window size = 991 individuals

(e) Window size = 2991 individuals

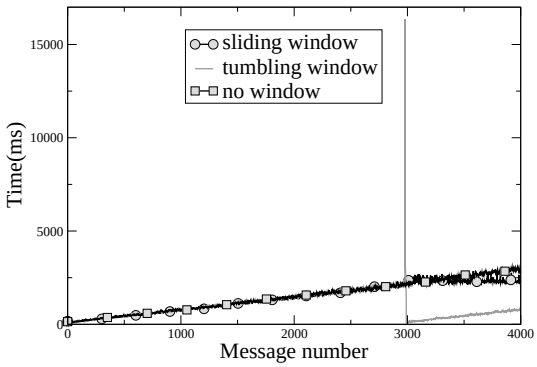
(f) Window size = 2991 individuals

Figure 5: Performance and number of statements for unit step sliding window of various sizes. Raw model refers to the initial model while inferred refers to the model after the reasoning procedure.



(a) Process time, window size=191 individuals

(b) Process time, window size=991 individuals



(c) Process time, window size=2991 individuals

Figure 6: Comparison of performances for the three methods for various window sizes.