SocEDA

Cloud based platform for large scale social aware EDA

ANR-10-SEGI-013

Document name:	Event Cloud : State-of-the-Art and Requirements.
Document version:	1.0
Task code:	D2.3.1
Deliverable code:	
WP Leader (organisation):	I3S
Deliverable Leader (organisation):	LIG
Authors (organisations):	I3S ; LIG
Date of first version:	T0+12

Change control

Changes	Author / Entity	Code of version
Initial draft on state of the art	I3S	V0.1
Initial draft on requirements	I3S	V0.2
Extension regarding state of the art	LIG	V0.3
Estension regarding requirements (robustness, load sharing)	LIG	V0.4
Introduction and Conclusion	LIG	V0.5

Table of Contents

1. Introduction

The federated middleware layer developed within the SocEDA project encompasses different layers: Event cloud, DCEP, DSB, etc. In this document, we focus on the event cloud layer. In particular, we describe the requirements of this layer and relevant state of the art.

The role of the event cloud layer is central because each event gets propagated and stored through the event cloud layer. More concretely, the event cloud layer can be seen as a federation of different and independent event clouds. In each cloud, events get stored and each cloud is the place where event brokering is achieved given existing event subscriptions.

The event clould layer is composed of two main sub-components: the event storage component and the event brokering component. In this deliverable, we describe the requirements of these two sub-components. We also present software engineering requirements.

The document is organized as follows: we start by a brief definition of the related work. We focus on two distinct categories of systems: P2P systems managing RDF data and publish/subscribe systems. The choice for presenting these systems is leaded by the fact that the SocEDA event cloud component will implement a **publish/subscribe system for RDF data implemented on top of a P2P system**. We then present the requirements of the event cloud layer. Finally, we conclude this document.

2. State of the art

This section is dedicated to the description of related works belonging to two categories: P2P systems managing RDF data and publish/subscribe systems.

2.1. P2P systems managing RDF data

The P2P communication model has drawn a lot of attention and has emerged as a powerful architecture to build large scale distributed applications. P2P architectures are classified into two main categories:

- **Unstructured overlay networks:** in such systems, there is no constraint on the P2P architecture: the overlay is built by establishing random links among nodes (*Gnutella RFC*, 2003). Despite their simplicity, these systems suffer from several issues, in particular, the fact that search times are longer.
- **Structured overlay networks:** in these systems, peers are organized in a well-defined geometric topology (e.g., ring, torus, etc.) and, compared to unstructured networks, they exhibit stronger guarantees in terms of search time and nodes' maintenance (Castro et al., 2005). By providing a *Distributed Hash Table (DHT)* abstraction, by the mean of *Put* and *Get* methods, they offer simple mechanisms to store and fetch data easily in a distributed environment. However, even if this abstraction is practical, it has its limits when it comes to *efficiently* supporting complex types of queries. The first wave of DHTs (Stoica et al., 2001; Ratnasamy et al., 2001) focused on scalability, routing performance and peer churn resilience. The main issue of these DHTs is that they only support *exact queries*, i.e., querying for data items matching a given key *(lookup(key)*). To overcome such limitation, a second wave of DHTs led to a set of P2P approaches such as (Aberer et al., 2003; Cai et al., 2003) having the capabilities to manage more complex queries such as range queries (Aberer et al., 2003) or multi-attribute queries (Cai et al., 2003).

Within the context of the SocEDA project, we will use a structured overlay network. We do thus focus on this kind of overlay in the reminder of this document.

Recently, motivated by the realization of the Semantic Web vision at the Internet level, solutions combining Semantic Web technologies and the P2P communication model have been proposed. As explained in (Staab & Stuckenschmidt, 2006), both technologies address the same need but at different levels. The Semantic Web allows users to model, manipulate and query knowledge, represented using the Ressource Description Framework (RDF) data model (Resource Description Framework 1999)), at a conceptual level with the intent to exchange it. P2P technologies, on the other hand, enable users to share information using the decentralized organization principle. P2P systems have been recognized as a key communication model to build scalable platforms for distributed applications such as file sharing (e.g., Gnutella (*Gnutella RFC*, 2003)) or distributed computing (e.g.,SETI@home (*SETI@home*, n.d.)).

Several works have thus focused on combining P2P systems and the RDF data model (see survey (Filali et al., n.d.) for further details). In addition to the basic challenges related to such large scale infrastructures (network partition, network maintenance, etc.), enabling complex querying of RDF data on top of such infrastructure requires advanced techniques for data indexing and query processing algorithms. Regarding data indexing, several approaches, e.g., RDFPeers (Cai & Frank, 2004), Atlas (Koubarakis et al., 2006), YARS (Harth & Decker, 2005), RDFCube (Matono et al., 2007), Battré *et al.* in (Battré et al., 2006), while based on different overlay topologies, share almost the same data indexing model by *hashing* the RDF triple elements. The main advantage of this indexing strategy is that triples with the same subject, object and predicate are stored on the same node and thus can be searched locally and without needing to be collected from all data sources. However, individual nodes, responsible for overly popular triples (e.g., rdf:type, dc:title), can be easily overloaded resulting in poor performance. One possibility to address this issue is to use multiple hash functions to ensure a better load distribution as proposed in (Ratnasamy et al., 2001) and recently in (Mu et al., 2009).

The second category of RDF-based P2P approaches harness the semantic of RDF data either to build a “semantic” layer on top of the P2P overlay (e.g., (Cudré-Mauroux et al., 2007; Karnstedt et al., 2008)) or to adopt a semantic clustering approach and organize the P2P layer as function of the semantic of the stored data (e.g., Edutella (Nejdl et al., 2003), DSS (Gu et al., 2007), S-RDF (Zhou et al., 2009), Bibster (J. Broekstra et al., 2004)). Others have extended the basic RDF data model by adding the “context” concept as in YARS (Harth & Decker, 2005) and PAGE (Della Valle et al., 2006). By taking into consideration data semantics in the data indexing phase, these approaches try to improve data lookup. However, this requires an additional effort to maintain the mapping between the semantic and the overlay levels.

Regarding robustness, most of the presented works aim at adding data *availability* feature to the RDF storage infrastructure through *data replication*. However, data replication further raises several issues. Thereof, three main challenges have to be taken into consideration: which data items have to be replicated; where to place replicas and finally how to keep them consistent. In P2P systems there has been a lot of work on managing data replication. In (Ktari et al., 2007), Ktari *et al.* investigated the performance of several replication strategies under DHTs systems including neighbour replication, path replication and multi-key replication. As argued in (Ktari et al., 2007), the data replication strategy can have a significant impact on the system performance. Further effort may go into exploring more “dynamic” and adaptive replication approaches as function of data popularity or average peer online probability. Although the replication techniques increase the data availability, they come not only at the expense of more storage space but can also affect the *data consistency* (e.g., concurrent update for the same triple). Moreover, the data inconsistency issue becomes even more intricate under the partitioning of the P2P network. Thus, an update operation might not address all replicas as a node storing a replica can be offline during the update process. Therefore, trade-offs are made between high data availability, data consistency and partition-tolerance. Brewer brought all these tradeoffs together and presented the CAP theorem (Gilbert & Lynch, 2002) which states that with Consistency, Availability, and Partition-tolerance, we can only ensure two out of these three properties. Recognizing which of the “CAP” properties the application really needs is a fundamental step in building a successful distributed, scalable, highly reliable system.

2.2. Publish/Subscribe systems

The goal of Publish/Subscribe systems is to allow processing/managing long-standing queries. In SocEDA, subscribers will express their interests in specific events using a SPARQL query and will be notified when there will be notifications matching their interest. The Publish/Subscribe paradigm (see (Eugster et al., 2003) for a more detailed survey) introduces query decoupling in space, time and synchronization between participants. In a publish/subscribe system, subscribers, also called consumers, can express their interests in an event or a pattern of events, and be notified of any generated event by the publishers (producers) that matches those interests. The events are propagated asynchronously to all subscribers. Therefore, the overall system is responsible for matching the events to the subscriptions and for the delivery of those relevant events from the publishers to the subscribers which are distributed across a wide area network via Notifications.

The publish/subscribe communication paradigm is usually used to exchange information between applications, services and devices, for its ability to decouple communication participants. The publish/subscribe communication paradigm is frequently chosen over the request/response one to exchange information, mainly for its asynchronous nature and the loose coupling of communication participants (Rodriguez-Dominguez et al. 2010; Cugola and Jacobsen 2002). Anonymity and immediate communication that is supported in publish/subscribe systems (Eugster et al. 2003) constitute another advantage. The aim of these systems is to disseminate published information to a set of subscribers. Each subscriber receives only the information that matches their subscription. In dynamic and highly distributed environments, there are many information providers that if they were indiscriminately and constantly pushing information, they could flood any system. So, not all of this data is to be pushed to the consumers but only the relevant to the users'/services' preferences.

The consumers express their interests by subscribing to events of a given type. When a new event is published, the system compares it against all existing subscriptions and notifies interested parties. In the state of the art in publish/subscribe systems both operations are done explicitly (Berkovsky and Eytani 2005). Users define, at design time, the events they publish and manually inform the system about the events they are interested in. Publish/subscribe systems act as natural mediators between information providers and information consumers in distributed environments (Aguilera et al. 1999). Information providers publish information in a form of events, while information consumers subscribe to a particular category of events. The system is responsible to check the event against all current subscriptions and deliver it to the consumers, whose subscriptions match the event category.

Overall, the key components of a publish/subscribe system can be summarized into the following concepts:

Subscriptions: A subscription describes a set of notifications (i.e., events) a consumer is interested in. The goal behind the subscription process is that subscribers will receive notifications matching their interests from other peers in the network. Subscriptions are basically *filters*, which can range from simple Boolean-valued functions to the use of a complex query language. The expressiveness of the subscriptions in terms of filtering capabilities depends directly on the data model and the filter model used.

Notifications: In publish/subscribe systems, notifications can signify various types of events depending on the perspective. From a publisher perspective, *advertisements* are a special type of notification used to describe the kind of notification the publishers are willing to send. From a subscriber perspective, a notification is an event that matches the subscription(s) of consumers[1] . An event notification service is the mediator which is responsible for conveying notifications to subscribers. Several peers within the network can actively or passively participate in the dissemination of those notifications.

Publish/subscribe systems have several ways for identifying notifications that can be based either on a *topic* or the *content*. In the *topic-based* model, publishers annotate every event they generate with a string denoting a distinct topic. Generally, a topic is expressed as a rooted path in a tree of subjects. For instance, an online research literature database application (such as IEEE Xplore) could publish notifications of newly available articles from the *Semantic Web* research area under the subject "/Articles/Computer Science/Proceedings/Web/Semantic Web". This kind of topic will then be used by subscribers which will, upon subscription's generation, explicitly specify the topic they are interested in and for which they will receive every related notifications. The topic-based model is at the core of several systems such as Scribe (Castro et al., 2002), Sub-to-Sub (Voulgaris et al., 2006). A main limitation of this model lies in the fact

[1] In fully decentralized pub/sub systems, every node can perform the matching process.

that a subscriber could be interested only in a subset of events related to a given topic instead of all events. This comes from the tree-based classification which severely constrains the expressiveness of the model as it restricts notifications to be organized, using a single path in the tree. A tree-based topic hierarchy inhibits the usage of multiple super-topics for instance, even if some inner re-organizations are possible, this classification mechanism remains too rigid. On the other side, a *content-based* model is much more expressive since it allows the evaluation of filters on the whole content of the notifications. In other words, it is the data model and the applied predicates that exclusively determine the expressiveness of the filters. Subscribers express their interests by specifying predicates over the content of notifications they want to receive. These constraints can be more or less complex depending on the query types and operators that are offered by the system. Available query predicates range from simple comparisons, regular expressions, to conjunctions, disjunctions, and XPath expressions on XML.

Publish/Subscribe systems are the topic of many research works on orthogonal, non-functional, aspects. Let us cite for instance research works focusing on improving the "**security**" aspects of publish/subscribe systems: in (Wang et al. 2002), the authors analyze the security issues and requirements that arise in content-based publish subscribe systems. They mainly identify classical security problems (like authentication, integrity or confidentiality) and adapt them to the content-based publish/subscribe case. Other works focus on the "**confidentiality**" aspects of publish/subscribe systems. For instance, in (Raiciu & Rosenblum. 2006), the authors define the confidentiality issues in a formal model and propose solutions depending on the subscription and notification format. Several works are also devoted to "**subscription automation**" aspect. For instance, (Brenna et al. 2006) proposes an approach that shows how subscriptions in publish/subscribe systems can be automated with a novel application of information retrieval techniques. They propose the Reef framework that is geared toward monitoring Web browsing history and making recommendations to a user through a browser extension, in an automatic way. In

Finally, to conclude this section, we would like to cite a few works that combined P2P techniques (studied in the previous section) and publish/subscribe techniques that are the subject of this section. For instance, Meghdoot (Gupta et al., 2004) leverages a structured overlay network (called CAN). Meghdoot provides an efficient mechanism for disseminating events to subscribers but it has limitations regarding the query expressiveness it offers: Meghdoot does not support range subscriptions, and is confined to numerical attributes. Another interesting work is Vitis (Rahimian et al., 2011) , a novel publish/subscribe which groups nodes which have similar topic interests altogether and use mostly these nodes to convey notifications to subscribers. In doing so, only nodes that have some interests in the messages they route participate in the dissemination process. The architecture of Vitis is based on a ring-based structured overlay network, *à la* Chord (Stoica et al., 2001) where nodes exploit subscription similarities using a utility function, thus selecting best fitted neighbors which will form a navigable small world (Kleinberg, 2000) overlay network. Note that none of Meghdoot and Vitis are able to manage RDF data.

3. Requirements

This section is dedicated to the description of the requirements of the event storage and the event brokering components. We conclude this section by a description of software engineering requirements.

3.1. Requirements for the event storage

3.1.1. Functional requirements

Due to the increasing number of Web Services and the huge amount of data they produce, the event storage has to be largely distributed and scalable. Consequently, we decided to construct the event storage on top of a CAN structured peer-to-peer network. The RDF storage repository is implemented using a four-dimensional CAN overlay. The four dimensions of the CAN coordinate space represent respectively the subject, the predicate, the object and the graph value of the stored RDF quadruple. Technically minded, each peer indexes the quadruples within its zone by using a Jena RDF store, which is an actively maintained framework that provides API and mechanisms to write and to retrieve RDF data but also to serialize semantic data into several formats.

In addition to a Put/Get API (to populate “synchronously” the store with for example data from a given database), each event cloud will offer a Publish/Subscribe messaging pattern, which, thanks to the decoupling of subscribers and publishers allows greater scalability.

Regarding the communications with an event cloud, the entities (SocEDA components, Web Services, applications or users) that need to interact with an event cloud will use a proxy. A proxy is a light application that knows how to contact an event cloud (via the Trackers that serve as an entry point into the peer-to-peer network) and how to execute some operations. As Figure 1 depicts, to meet the platform needs, we decided to implement three proxies:

- PutGetProxy is used to insert RDF data and to retrieve RDF data by using SPARQL queries synchronously.
- PublishProxy is used to publish RDF data or Events asynchronously.

- SubscribeProxy is used to subscribe to some RDF data or Events by using a SPARQL query. It is also used to guarantee some properties (e.g. maximum throughput of notifications per time unit).

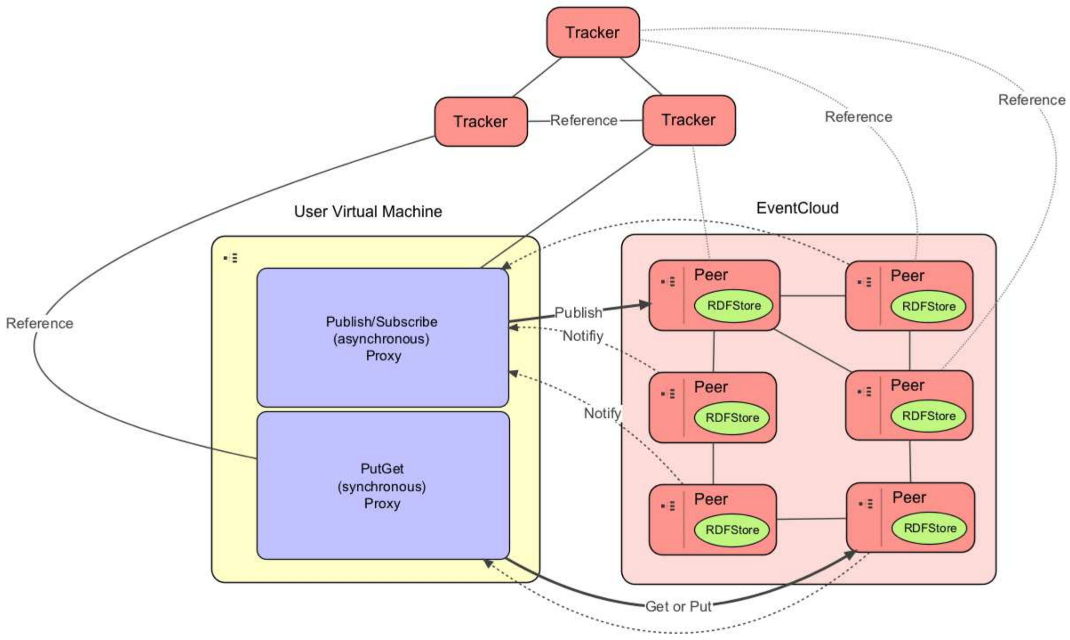

Figure 1: Proxies Interaction with an Event Cloud

3.1.2. Non-functional requirements

There are three requirements for the event storage component. First, the storage needs to be elastic to handle changing loads. Second, it needs to be robust, which means that events should be reliably stored. Third, it is necessary to fairly balance the load on the various peers forming the storage (and more precisely the CAN network used to implement them). In this section, we detail these two requirements.

<u>Elasticity</u>

The amount of information that will be handled by an event cloud is both unpredictable and fluctuating. Consequently, an event cloud has to be elastic. In contrary to an always-on infrastructure for which the institutions refrain to pay for, the idea is to rely on the notion of Cloud

Computing to scale horizontally by adding more nodes on demand and to release them as soon as possible. But also to scale vertically by offering the possibility to deploy several peers on the machines that are under loaded. Within the context of SocEDA, we will implement some existing heuristics to make the storage layer elastic.

Robustness

As explained in the previous section, the event storage is constructed on top of a structured peer-to-peer network, called Content Addressable Network (CAN). It is necessary to ensure that it is reliable. This can be achieved by replicating RDF tuples that are stored. This roughly consists in storing multiple copies of each RDF tuple on different nodes (called replicas). A replication protocol is in charge of ensuring that, at any time, each RDF tuple is replicated on a sufficiently large number of replicas (called the replica set). The replication protocol is also in charge of deciding where replicas should be localized.

In the past years, several replication strategies have been proposed in the context of peer-to-peer systems, and in particular Distributed Hash Tables. The basic replication strategy is quite simple: replicas are simply placed on nodes that are “close” (in the overlay network) to the node managing the RDF tuple. This strategy induces nevertheless a very high bandwidth consumption to migrate replicas whenever a node arrival or departure changes the architecture of the overlay network. A better strategy has been proposed that takes into account the availability of nodes. This strategy relies on availability prediction algorithms which drive the replication strategy to place and to migrate replicated data on the most available nodes.

Within the context of the SocEDA project, we would like to propose another strategy: we believe that it would be interesting to exploit the regularity of node connections. Indeed, recent studies have shown that diurnal availability patterns are commonly exhibited by real-world systems. The idea would be to create for each RDF tuple a candidate set containing nodes exhibiting regular connection patterns such that, with a high probability, there will always be k online nodes in the candidate set to replicate the given RDF tuple. That way, these are always the same set of nodes that will periodically belong to the replica set of a given RDF tuple. Nodes can thus cache data and drastically reduce the bandwidth usage.

Load sharing

Another requirement for the storage layer is to fairly share the load of RDF triples to be indexed and stored among nodes participating in the CAN network. Traditionally, peer-to-peer systems seek to equally balance the load on all nodes. In most systems, all nodes periodically perform the same set of actions. Yet, large-scale distributed systems are usually heterogeneous with respect to network capabilities such as bandwidth. We do thus believe that a blind load-balancing strategy, as usually implemented in peer-to-peer systems, might significantly hamper the performance of the system.

Within the context of the SocEDA platform, we would thus like to study mechanisms allowing to adapt the contribution of nodes in the CAN network according to their bandwidth and storage capabilities. This requires modifying the way the CAN network is used. It also requires mechanisms to dynamically assess the capability of a given node and to compare it to that of other belonging to the network.

3.2. Requirements for event brokering

3.2.1. Functional requirements

The event brokering layer implements a publish/subscribe mechanism. To build a publish/subscribe system, a notification classification scheme has to be defined. A notification classification scheme is composed of a data model and a filter model. Within the SocEDA project, we propose to build the data model on top of quadruples by associating each element of a quadruple to a dimension of a CAN structured peer-to-peer network. Thus, each quadruple represents a potential event that may be delivered to a subscriber. However, by using this data model, the number of elements contained by an event (quadruple) is limited. To overcome this drawback we propose to introduce the notion of compound event (CE): an event that is made of a non-limited number of quadruples. All the quadruples that belong to the same composite event are assumed to share the same graph value.

Regarding the filter model, we must have the possibility to formulate a subscription by associating some filter constraints to a quadruple (event), but also to a set of quadruples that belongs to the same compound event. This means that several events (quadruples) that are published at different times and that belong to a same compound event may participate to the matching of a subscription by using their common constraints.

Moreover, in order to guarantee fairness regarding a notification point of view, events should be delivered in a preserved order. This means that event sources will have to ensure the correct order of the messages when pushed into an event cloud. The dissemination of the events to the subscribers will then be processed by the Event Cloud, which will make its best to guarantee the same order even in the presence of failures. An additional requirement regarding event delivery order is that notifications for a suite of subsequently ordered publications (of a same source) that matches a given subscription reach the corresponding subscriber in the same order. This means that the matching of a given subscription by two consecutive published compound events generates notifications that are ordered in the same way.

3.2.2. Non-functional requirements

There are two requirements for the event brokering component. First, it needs to balance the load on the available nodes. Second, it needs to be robust.

<u>Load balancing</u>

The event brokering component should be able to balance the load. It is well-known that the distribution of RDF data elements is inherently skewed. To face this challenge, we first propose to study two different techniques: first, we can remove the prefix from each RDF tuple in order to mitigate the data distribution. The second technique is to force new peers to join in priority an overloaded peer in order to lower its load. These two solutions should help to better share the load across nodes in the overlay. We then propose to investigate load balancing techniques that have been proposed in content-based publish/subscribe systems: these techniques rely on a load estimation algorithms. Some techniques have also been proposed that work by reducing the spatial extent of a zone managed by a peer in order to reduce the number of subscriptions it handles. Finally, we would like to mention the fact that in a CAN overlay network, we are in a presence of a very rich, self-organized network of brokers. Routing paths between any two peers in a CAN are numerous, which can give opportunities to allow messages to use alternate paths to avoid congested zones. If necessary, we will investigate a solution based on this observation.

Fault-tolerance

It is necessary to make sure that the event brokering component will be resilient to failures. This means that events should be delivered in the correct order, despite the failure of part of the nodes composing the event brokering component. A general approach to fault-tolerance is to use “state machine replication”. State machine replication is a software technique for tolerating failures using commodity hardware. The critical service to be made fault- tolerant is modelled by a state machine. Several, possibly different, copies of the state machine are then deployed on different nodes. Clients of the service access the replicas through a state machine replication protocol which ensures that, despite concurrency and failures, replicas perform client requests in the same order. In a replicated pub/sub system for instance, all replicas must perform the same client requests (subscriptions, publications) in the same order.

Two objectives underlie the design and implementation of a state machine replication protocol: *robustness* and *performance.* Robustness conveys the ability to ensure availability (liveness) and one-copy semantics (safety) despite failures and asynchrony. On the other hand, performance measures the time it takes to respond to a request (latency) and the number of requests that can be processed per time unit (throughput). The most robust replication protocols are those tolerating arbitrary failures of replicas, and not only straight failures of brokers. These protocols, called BFT (Byzantine Fault-Tolerant) protocols are those we should consider within the context of the SocEDA project. A number of BFT protocols exist. Nevertheless, existing BFT protocols have two main drawbacks. First, most existing BFT protocols have not been devised for wide-area networks. We would like to propose BFT protocols optimized for wide-area communications. Such protocols must provide both high throughput under high client load and low latency under low client load. They must also be able to face changing network conditions and varying client load. Finally, they must be able to leverage asymmetric network link performance. Second, existing BFT protocols do only tolerate faults from a fraction of the nodes (typically, one third of the nodes). Within the context of the SocEDA project where the event brokering component may be deployed also on machines gained from volunteer peer-to-peer and cloud systems, we believe that it is important to consider selfish behaviours. Indeed, nodes participating to the event brokering component can behave selfishly, i.e. they can try to maximise their benefit, while reducing their contribution. From an external viewpoint, selfish nodes behave like dysfunctional nodes. To ensure the efficiency of the publish/subscribe system, it is thus necessary to design techniques forcing selfish nodes to behave correctly.

3.3. Software engineering requirements

Previous sections highlighted challenging requirements in the way events must be handled within an event cloud. We have seen that the two involved components (event storage and event brokering) have both functionl and non-functional requirements. Moreover, it is possible that these two components need to be adapted at runtime, e.g. if the operating conditions change, or if the user has new requirements. Consequently, we believe that it is of primary importance to use a software engineering technique allowing both static and dynamic configurations of components. Within the context of the SocEDA project, we argue that the right technique to use is the component-based technique. Components have already proven useful when developing base components of the SocEDA stack (e.g. the Petals DSB). We would like to use them to develop the two components described in this deliverable: the event storage and the event brokering components.

According to this guideline of applying a component-oriented approach to program the event cloud, each peer of the event cloud will be represented as a distributed software component, equipped with the necessary control part in order to be able to possibly act, i.e. reconfigure the peer at runtime, control its QoS, its load, etc. In order to fully leverage the possibility offered by component-based programming models, a requirement will be to distinguish (at the code level) the functional layer from the control (also named non-functional) layer.

4. Conclusion

In this deliverable, we have described the requirements of the event cloud layer. This layer is composed of two components: the event storage component and the event brokering component. For each component, we have described both functional and non-functional requirements. To better understand the challenges, we have started with a brief desription of related works in the two main areas addressed by the event cloud layer: P2P systems managing RDF data and publish/subscribe systems. Finally, we have concluded this deliverable with a short description about software engineering requirements.

References

Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M. & Schmidt, R. (2003), 'P-Grid: a self-organizing structured P2P system', ACM SIGMOD Record 32(3), 33.

Aguilera, MK, Strom, RE, Sturman, DC, Astley, M & Chandra TD 1999, 'Matching events in a content-based subscription system', Symposium on Principles of Distributed Computing, Atlanta.

Battré, D., Heine, F., Höing, A. & Kao, O. 2006, 'On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT Based RDF Stores', in DBISP2P, pp. 343–354.

Berkovsky, S & Eytani, Y 2005, 'Semantic Platform for Context-Aware Publish/Subscribe M-Commerce', Proceedings of the SAINT Workshops, Trento Italy, pp. 188-191.

Brenna, L, Gurrin, C, Johansen, D & Zagorodnov, D 2006, 'Automatic subscriptions in publish-subscribe systems', Proceedings of the 26th IEEE Intl Conf. on Distributed Computing Systems (ICDCS) Workshops: Distributed Event-Based Systems (DEBS), pp. 23-23.

Broekstra, J, Ehrig, M, Haase, P, van Harmelen, F, Menken, M, Mika, P, Schnizler, B & Siebes R 2004, 'Bibster - A Semantics-Based Bibliographic Peer-to-Peer System', in The Second Workshop on Semantics in Peer-to-Peer and Grid Computing (SEMPGRID), New York.

Cai, M., Frank, M., Chen, J. & Szekely, P. 2003, 'MAAN: A multi-attribute Addressable Network for Grid Information Services', in Journal of Grid Computing, Vol. 2.

Cai, M., Frank, M. R., Yan, B. & MacGregor, R. M. 2004, 'A subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management', J. Web Sem. 2(2), 109–130.

Castro, M., Costa, M. & Rowstron, A. 2005, 'Debunking Some Myths about Structured and Unstructured Overlays', in Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation (NSDI), USENIX Association, pp. 85–98.

Castro, M., Druschel, P., Kermarrec, A. & Rowstron, A. 2002, 'SCRIBE: A large-scale and decentralized application-level multicast infrastructure', IEEE Journal on Selected Areas in Communications 20(8), 1489–1499.

Cudré-Mauroux, P., Agarwal, S. & Aberer, K. 2007, ‘Gridvine: An infrastructure for peer information management’, IEEE Internet Computing 11, 36–44.

Cugola, G & Jacobsen, HA 2002, ‘Using publish/subscribe middleware for mobile systems’, SIGMOBILE Mob. Comput. Commun. Rev., vol. 6, no. 4, pp. 25–33.

Della Valle, E., Turati, A. & Ghioni, A. 2006, ‘PAGE: A distributed infrastructure for fostering RDF-based interoperability’, in Distributed Applications and Interoperable Systems (DAIS), Springer, pp. 347–353.

Eugster, PT, Felber, PA & Kermarrec, AM 2003, ‘The many faces of publish/subscribe’, ACM Computing Surveys, vol. 35, pp. 114-131, 2003.

Filali, I., Bongiovanni, F., Huet, F. & Baude, F. (n.d.), RDF Data Indexing and Retrieval: A survey of Peer-to-Peer based solutions, Technical report. URL: http://hal.archives-ouvertes.fr/inria-00540314/en/

Gilbert, S. & Lynch, N. 2002, ‘Brewer’s Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services’, in ACM SIGACT News, p. 2002.

Gnutella RFC 2003, http://rfc-gnutella.sourceforge.net/.

Gu, T., Pung, H. K. & Zhang, D. 2007, ‘Information Retrieval in Schema-based P2P Systems Using One-dimensional Semantic Space’, Computer Networks 51(16), 4543–4560.

Gupta A., Sahin O. D., Agrawal D., et El Abbadi A., ‘Meghdoot: Content-Based Publish/Subscribe over P2P Networks’, in Middleware 2004, 2004, p. 254-273.

Harth, A. & Decker, S. 2005, ‘Optimized Index Structures for Querying RDF from the Web’, in Proceedings of the Third Latin American Web Congress (LA-WEB), IEEE Computer Society, Washington, DC, USA, p. 71.

Karnstedt, M., Sattler, K.-U., Hauswirth, M. & Schmidt, R. 2008, ‘A DHT-based Infrastructure for Ad-hoc Integration and Querying of Semantic Data’, in Proceedings of the 2008 international symposium on Database engineering and applications (IDEAS), ACM, New York, NY, USA, pp. 19–28.

Kleinberg J., ‘The small-world phenomenon: An algorithmic perspective’, in Annual ACM symposium on theory of computing, 2000, vol. 32, p. 163–170.

Koubarakis, M., Miliaraki, I., Kaoudi, Z., Magiridou, M. & Papadakis-Pesaresi, A. 2006, ‘Semantic Grid Resource Discovery using DHTs in Atlas’, in Proceedings of 3rd GGF Semantic Grid Workshop, Athens, Greece.

Ktari, S, Zoubert, M, Hecker, A & Labiod, H 2007, ‘Performance Evaluation of Replication Strategies in DHTs Under Churn’, in Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia (MUM), ACM, New York, NY, USA, pp. 90–97.

Matono, A., Pahlevi, S. & Ko jima, I. 2007, ‘RDFCube: A P2P-Based Three-Dimensional Index for Structural Joins on Distributed Triple Stores’, Databases, Information Systems, and Peer-to-Peer Computing pp. 323–330.

Mu, Y., Yu, C., Ma, T., Zhang, C., Zheng, W. & Zhang, X. 2009, ‘Dynamic Load Balancing With Multiple Hash Functions in Structured P2P Systems’, In Proceedings of the 5th International Conference on Wireless communications, networking and mobile computing (WiCOM), IEEE Press, Piscataway, NJ, USA, pp. 5364–5367.

Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I. & Löser, A. 2003, Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks, in ‘Proceedings of the 12th international conference on World Wide Web’, ACM, pp. 536–543.

Raiciu, C. & Rosenblum., D. S. (2006), Enabling condentiality in content-based publish/subscribe infrastructures, in `Securecomm and Workshops', p. 1-11.

Rahimian F., Girdzijauska S., Payberah A., Haridi S., Vitis: A Gossip-based Hybrid Overlay for Internet-scale Publish. In IPDPS 2011, 16-20 May 2011, Anchorage, Alaska, USA.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. & Shenker, S. 2001, ‘A Scalable Content-Addressable Network’, in Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), ACM, pp. 161–172.

Resource Description Framework (1999). http://www.w3.org/RDF/

Rodriguez-Dominguez, C, Benghazi, K & Noguera M 2010, ‘Redefinable events for dynamic reconfiguration of communications in ubiquitous computing', Proceedings of the First International Workshop on Data Dissemination for Large Scale Complex Critical Infrastructures (DD4LCCI '10), Valencia, Spain, pp. 17-22.

SETI@home (n.d.). http://setiathome.ssl.berkeley.edu/.

Staab, S. & Stuckenschmidt, H. 2006, Semantic Web and Peer-to-Peer, Springer.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. & Balakrishnan, H. 2001, 'Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications', in Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), ACM, New York, NY, USA, pp. 149–160.

Voulgaris, S., Rivière, E., Kermarrec, A.-M. & Steen, M. V. 2006, 'Sub-2- sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks', in Proceedings of the fifth International Workshop on Peer-to-Peer Systems (IPTPS).

Wang, C., Carzaniga, A., Evans, D. & Wolf, A. (2002), Security issues and requirements for Internet-scale publish-subscribe systems, in `System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on', IEEE, pp. 3940-3947.

Zhou, J, Hall, W & Roure, DD 2009, 'Building a Distributed Infrastructure for Scalable Triple Stores', Journal of Computer Science and Technology, vol. 24, no. 3, pp. 447–462.