SensBution: A Rule-Based Peer-to-Peer Approach for Sensor-Based Infrastructures

Tom Gross, Thilo Paul-Stueve, Tsvetomira Palakarska
Faculty of Media
Bauhaus-University Weimar
Bauhausstr. 11
99423 Weimar, Germany
<firstname.lastname>(at)medien.uni-weimar.de

Abstract

Ubiquitous environments facilitate the fast and easy access to users' physical environments. Ubiquitous environments typically capture users' presence and activities in a room with a multitude of sensors, process and infer on the captured data, and adapt the environment accordingly. In order to create working ubiquitous environments developers need adequate platforms that allow them to focus on novel concepts and implementations for the interaction of the users with their ubiquitous environment. SensBution provides generic and flexible support for developers of ubiquitous environments through a combined approach that is event-based and rule-based, and offers client-server and peer-to-peer structures. In this paper we present SensBution. We present the concept, base technology used, and implementation of SensBution. And finally we compare SensBution to related work.

1 Introduction

Ubiquitous environments facilitate the fast and easy access to users' physical environments. They go beyond the traditional interaction between users and the computer based on graphical user interfaces, mice and keyboards. Ubiquitous environments typically capture users' presence and activities in a room with a multitude of sensors, process and infer on the captured data, and adapt the environment accordingly.

In order to create working ubiquitous environments developers need adequate platforms that allow them to focus on novel concepts and implementations for the interaction of the users with their ubiquitous environment by relieving them from technical details.

Several approaches for platforms are available, but they typically have their trade-offs: the first central trade-off is between event-based and rule-based approaches; and the second trade-off is between client-server and peer-to-peer approaches. Event-based approaches allow the fast reaction to captured events, but require detailed and thorough knowledge and handling of technological details; whereas, rule-based approaches provide a better abstraction from technical details and queries of higher complexity, but entail delays due to the processing of the rules. Client-server based approaches support the fast access to sensors and sensor event data, but require detailed knowledge and handling of technical details of the infrastructure such as IP-addresses and port numbers of servers; whereas, peer-to-peer approaches provide robust structures and facilitate communication through firewalls without knowledge of the underlying network architecture, but can become slow because of high protocol and negotiation overheads.

SensBution provides generic and flexible support for developers of ubiquitous environments through a combined approach that is event-based and rule-based, and offers client-server and peer-to-peer structures. It has concepts of sensor-based infrastructures such as adapters for connecting sensors, inference engines for processing data, and gateways for connection actuators. It offers:

- fast replies to simple event-based queries
- complex replies to rule-based queries
- integration into client-server networks
- integration into peer-to-peer networks

We want to motivate the need for such a rule-based peer-to-peer approach in a little scenario. Imagine a typical day of a student Alice. Alice is in a hurry, she must go to the laboratory, but last time she was there, it was crowded, another time it was occupied and she could not do her work. SensBution provides a client application that gives information about the environment state of a certain location. The particular strength is that it allows to formulate queries in a natural language format (e.g., room states such as party, working atmosphere; mysterious; unpleasant; warm, cold; bright, dark; silent, loud; free, occupied; untroubled and hectic).

In this paper we present the concept and implementation of SensBution. In the next section we present the SensBution concept. We then describe the base technology used, and give details on the SensBution implementation of the rules and the peer-to-peer concepts. And we compare SensBution to related work.

2 SensBution Concept

In this section we give an overview of the concepts of SensBution. We introduce inferencing rules and peer-to-peer networks and then describe the rule-based query processing, the peers, and the flow of information.

An *inference rule* is a function consisting of one or more premises and a conclusion. It determines the logical value of a conclusion based on the logical value of the premises. If the premises are false, the conclusion is false, too. Inference rules have the following schema:

```
If <premise> Then <conclusion>
```

SensBution uses inference rules to abstract from the access to sensor event data. To deliver results, specialised rules for each type of query evaluate assigned information by matching them to the sensor event data stored in SensBution. The result is sensor event data that matches the specified information. In contrast to database queries no information about the internal structure of the database is needed.

Peer-to-peer networks are ad-hoc overlay-networks with a decentralised topology. They abstract from the underlying network structure and every peer in the network acts autonomously. A peer may provide services or use services provided by other peers in the network. A peer needs to be stable and flexible in its communication with other peers, especially since in peer-to-peer networks any other peer can cease its service.

SensBution integrates the peer-to-peer network concept to couple the single instances of the platform. The instances are able to exchange messages without needing to know about the technical details of the other platforms such as IP addresses of the host computers or communication ports.

The inferencing rules and the peer-to-peer concept are essential parts of rule-based query processing, the peers, and the flow of information of SensBution, which we describe subsequently.

2.1 Rule-Based Query Processing

By integrating rule-based inferencing into SensBution, it is possible to access sensor event values by providing parameters that describe the content of interest with specifications like location or type of an event. Technical details about the sensor that captured the event, sensor name, or sensor identifier are not necessary to access the sensor event data when using derivation rules to access data.

SensBution extracts sensor information from the existing sensor platform Sens-ation [8] to derive further information concerning the environment state. Light, temperature, noise, movement, and occupation values of the respective sensors measurements are merged into the rule structure to infer conclusion to a certain query. Thus a human-language query such as: ‘Is it currently cold in the student laboratory in Hausknechtstraße 7?’ can be formulated, and answered, through a simple rule:

```
IF the average value of a temperature sensor
  at a date and location is less than or
  equal to the threshold value
  of the temperature
  at the date and location
THEN it is cold.
```

The query contains all the data that represents the rule-based query process. In its simplest form this is one prerequisite and one fact. The query has a unique identification to make it addressable in the whole network, and a due date. The due date describes the date on which the user requires the answer by the system. The minimum due date is the time required by a SensBution environment to process the answer. Beside normal queries containing the data for one rule-based query process, there are queries representing more complex inference rules. They initiate the processing of up to three different subqueries.

The answer container includes all results of the rule-based inferencing and has the same identification as the query that evoked the generation of the answer to allow to reference the answer with the identification of that query.

2.2 SensBution Peers

Peers are the central components of SensBution. They consist of a broad range of subsystems and components described below. Figure 1 depicts the details of a single SensBution peer and shows its communication with other SensBution peers, clients, subscribers, and sensors.

Each SensBution peer has a *Gateway* subsystem responsible for the request and response handling over different kinds of protocols and interfaces (i.e., a peer-to-peer gateway, and client-server gateways for Web Services, remote method invocation, XML-RPC, common gateway interface, hypertext transfer protocol, and sockets). The Gateway receives service requests and forwards them via the GatewayHandler to the responsible subsystems that deliver response. The subsystems reply to the Gateway, and the Gateway in turn delivers the response to the service requester.

The *Gateway P2P* subsystem is responsible for the distribution of the queries to and the processing of the incoming answers from other SensBution peers. There is no central routing instance in the SensBution peer network that is able to direct queries to peers who are able to deliver answers. Therefore, queries are distributed according to the one-to-many or broadcast pattern. Every SensBution peer receives the queries of all other SensBution peers in the same peer group. The Gateway P2P subsystem delivers answer messages to the querying peer following the one-to-one pattern since this time the addressee is known.

The *RuleInferencing* subsystem is responsible for the handling and processing of rule-based queries. It has three components: the QueryHandler, the QueryProcessing, and the Rulebase. The *QueryHandler* component is responsible for the handling of queries incoming from the Gateway subsystem. In the case of a peer request—that is, if the query comes from another peer—the QueryHandler simply forwards the incoming query to *QueryProcessing* and

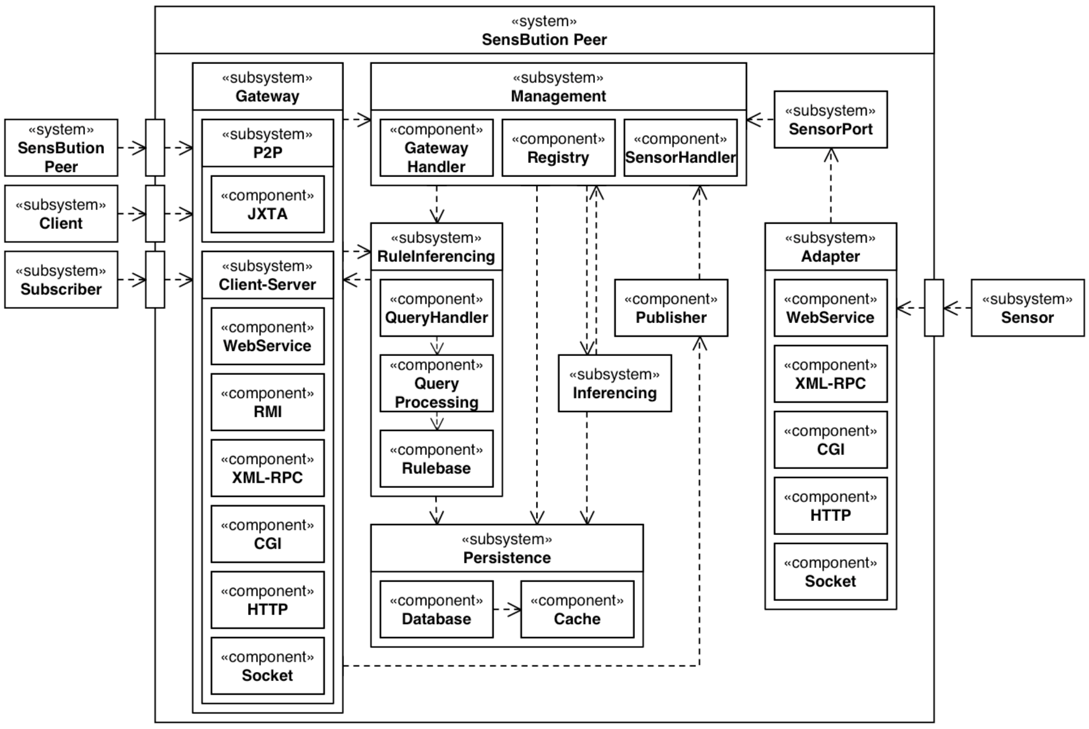

Figure 1. Components of a SensBution peer.

immediately gets the results. In the case of a client request—that is, if the query comes from a client—the QueryHandler first forwards the incoming query to Gateway P2P subsystem for distribution in the peer-to-peer network, and then forwards the query to QueryProcessing. It also receives the results from the QueryProcessing immediately and fetches the results from the peer-to-peer network via the Gateway P2P. In both cases the results are delivered to the requesting Gateway.

The QueryHandler incorporates a query scheduler that stores all incoming queries and takes care of their punctual evaluation by the appropriate subsystem. The QueryProcessing component extracts all parts of received queries and forwards them to the Rulebase for evaluation. The result received by the Rulebase is handed back to the QueryHandler. The Rulebase maintains all rules and supplies the derivation rules delivering the query results.

The remaining components have already been implemented in the SensBase infrastructure. We only provide a brief overview here; details can be found in [8]. The Management Subsystem comprises the GatewayHandler, the Registry, and the SensorHandler component. The GatewayHandler processes requests coming from different gateway components. The Registry maintains information about the resources of the platform, such as active sensors, or locations. The SensorHandler processes the sensor input received via the SensorPort. The SensorPort component receives all incoming sensor events via the adapters and forwards them to the Management subsystem. The Adapter subsystem allows sensors to connect to SensBution and submit sensor events using Web Services, XML-RPC, a common gateway interface, hypertext transfer protocol, or a socket connection. The Publisher component provides an advanced publisher-subscriber mechanism accessible through the RMI component of the Gateway subsystem. The Inferencing component provides an interface for inference engines that deduce higher-level information from sensor event data. The Persistence subsystem comprises the Database and a Cache component, the long and short term data storage of SensBution. It provides the Rulebase with the necessary sensor event data.

2.3 Peer-to-Peer Flow of Information

In SensBution the information flow in each peer and its RuleInferencing subsystem depends on the type of request.

In the case of a client request a contact SensBution peer receives a query of service-requesting client through the XML-RPC Gateway (cf. Figure 2). The Gateway forwards the query through the GatewayHandler to the QueryHandler. The QueryHandler distributes the query to the peer-to-peer network using the JXTA Gateway. The JXTA Gateway sends the query to all reachable SensBution peers, the remote peers. Each remote peer receives the query—the peer request—through the JXTA Gateway (cf. Figure 3). Its JXTA Gateway forwards the query to the QueryHandler. The QueryHandler sends the query to the QueryProcessing component to evaluate the query. The QueryProcessing component submits the query variables to the Rulebase, which accesses the event

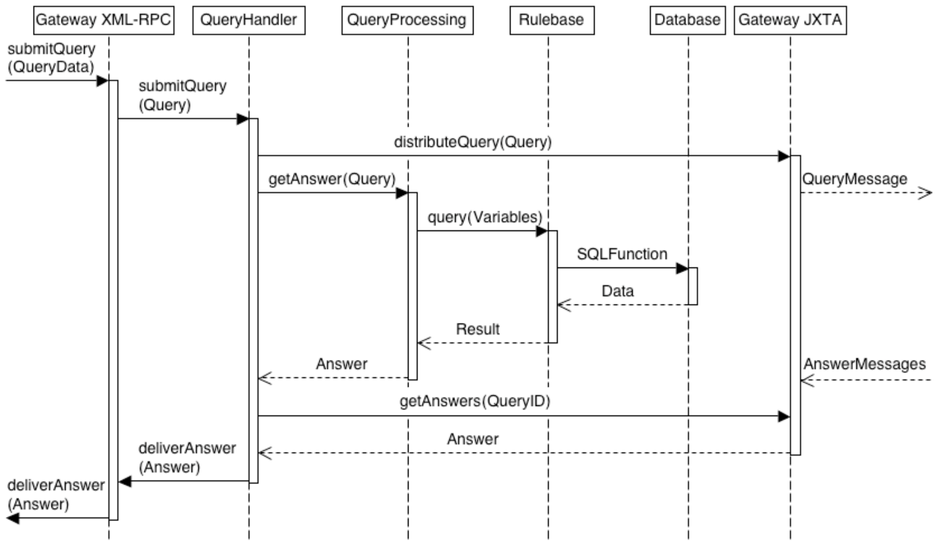

Figure 2. Processing of a client's query in the contact SensBution peer.

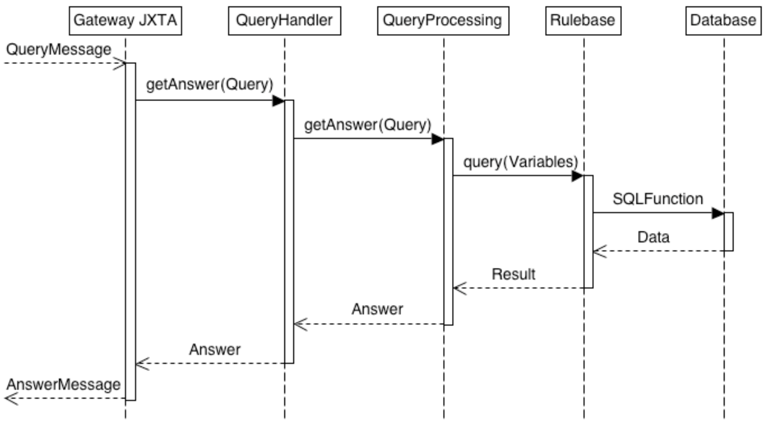

Figure 3. Processing the contact SensBution peer's query in the remote SensBution peers.

database through inferencing rules. The QueryProcessing component receives the result and hands the answer back to the QueryHandler, which sends the answer back to the querying contact SensBution peer through the JXTA Gateway. The JXTA Gateway of the contact SensBution peer receives and stores the answers. The QueryHandler calls the QueryProcessing component to receive an answer of the local instance. It contacts the JXTA Gateway to receive the stored answers of the remote SensBution peers. It then composes the result out of the local answers and the answers of the remote peers and sends it to the XML-RPC Gateway. The XML-RPC Gateway finally delivers the answer to the client.

3 Base Technology

The concept of SensBution was implemented based on the Mandarax framework, and the JXTA protocol. We briefly describe them below.

3.1 Mandarax

Mandarax is a Java framework for derivation rules and an implementation of a rule engine [5]. It facilitates the building, management, and querying of rule bases. It separates the processing logic (i.e., the rules) from the data (i.e., the facts). A rule engine applies the rules to the facts. It supports registration, rule classification, and management; it verifies the consistence of formal rules, and infers rules based on other rules.

The theoretical basis of Mandarax is first order logic. It provides a reference implementation for all logical objects—rules, facts, and terms—as interface and a logical factory object to create instances of them.

The prerequisites and the conclusion are called facts; they consist of predicates and terms. Java operators and methods can be used as predicates. The predicate takes the role as relation between entities and associates terms of a certain type. There are three different kinds of terms: constant terms that are represented by a concrete object instance; variable terms that are replaceable by concrete instance; and complex terms that can be computed by the application of a function or from other terms.

Facts and rules are called clauses. The rule system uses a set of clauses to represent knowledge. Knowledge bases are the containers for the clause sets; they can retrieve knowledge from it.

The Mandarax inference engine matches facts—the data—against rules, to infer conclusions.

The logical unit of the rule engine is the inference engine: an abstract machine that resolves facts and answers queries. The inference engine implements Prolog [4] algorithms for the logical derivation and uses the semantic of the Java object model to simplify the term structure. A query is a fact with variable terms. The inference engine evaluates queries. It requires queries and a knowledge base as input and returns a result set as response. Processing a query here means finding substitutions for the variable term within the query. The inference engine uses backward reasoning.

3.2 JXTA

The JXTA [3, 7] protocols specify an open-source peer-to-peer network platform. The provided set of XML-based protocols allow for discovery of other peers and their services, communication between the peers, and the formation of sub-groups of peers in the peer-to-peer network. The JXTA protocols are implemented on the Java 2 platform Standard Edition and Micro Edition, in C, C++, and C#.

A JXTA Peer is defined as an entity that implements the JXTA protocols required to act as a peer in the JXTA peer-to-peer network. JXTA Messages are the containers for information exchanged between the peers. JXTA peer groups provide boundaries for the visibility of peers and the distribution of messages.

The JXTA Abstraction Layer (JAL)[2] provides an easy to use abstract programmers interface to the JXTA protocols in the form of Java libraries.

4 SensBution Implementation

In the following we describe the implementation of the above concepts in the SensBution infrastructure. The SensBution infrastructure is implemented with the Java 2 Standard Edition 5.0 Platform [14]; combined with Mandarax version 3.3.2; and with JXTA 2.2.1, and JAL version 1.4.

Subsequently we describe the implementation of the rule-based query processing, and the implementation of the peer-to-peer concept.

4.1 Rule-Based Query Processing

The implementation of the rule-based query processing has two major parts: the `QueryHandler` implementing the QueryHandler concept, and the `Kowledgebase` implementing the QueryProcessing and the Rulebase concepts.

QueryHandler

The `QueryHandler` forwards incoming queries from the `GatewayXMLRPC` to the `GatewayJXTA` for distribution in peer-to-peer network by calling its `distributeQuery` method. Furthermore, the query is added to the `PriorityQueue` of the `QueryHandler`. The `PriorityQueue` stores the data for the query scheduler sorted by the query objects' due date. The scheduler is a thread that continuously compares the due date of the first `RBQuery` object in the `PriorityQueue` to the actual date. If the `RBQuery` object's due date is smaller or equal to the actual time, the `RBQuery` object is evaluated by calling the `processQuery` function of the `RuleInferencing`. The `QueryHandler` then fetches the remote answers from the peer-to-peer network by calling the `getAnswers` function of the `GatewayJXTA`, merges the local and remote `RBAnswer` objects, and calls the `deliverAnswer` method of the `GatewayXMLRPC`. Remote query calls received from the `GatewayJXTA` are processed immediately.

The `QueryHandler` thread identifies complex queries by their marker and processes them as a set of subqueries. When all answers of the subqueries are received, the complex answer is computed.

Knowlegebase

The `Knowledgebase` class implements the QueryProcessing and the Rulebase concepts. The `processQuery` function allows the submission of queries. It returns an answer, which is computed by specialised query processing methods that use the `rulebase`. The queries are processed by evaluating inference rules of the `rulebase` with variables. Figure 4 shows an example of a specialised processing method.

```
public RBAnswer processStateInformationQuery
 (String id, String pred, String sensorType,
  String date, String location) {

  RBAnswer result =
    new RBAnswer("No Response found!", id);
  String answer = " ";
  ResultSet rs = null;
  Class[12] struct = {
    String.class, String.class, String.class,
    String.class};
  VariableTerm queryVariable =
    (VariableTerm) lfs.variable(
      "sensor", String.class);
  Predicate p =
    new SimplePredicate(pred, struct);
  Query query =
    lfs.query(lfs.fact(p, sensorType, date,
    location, queryVariable),
    "get sensor state for " + sensorType);
  try {
    rs =
      ie.query(query, kb, InferenceEngine.ONE,
      InferenceEngine.BUBBLE_EXCEPTIONS);
    if (rs == null) {
      System.out.println
        (KnowledgeBase.class.getName() + " " +
        "Result Set is empty");
      return result;
    }
    while (rs.next()) {
      answer =
        (String) rs.getResult(queryVariable);
      result = new RBAnswer(answer,id);
    }
  } catch (InferenceException iee) {
    System.out.println
      (KnowledgeBase.class.getName() + " " +
      "Cannot compute sensor value for " +
      sensorType + "\n" + iee.getMessage());
    iee.printStackTrace();
  }
  return result;
}
```

Figure 4. Example of a SensBution method processing a location state query.

Inference rules build up the `rulebase` of the `Knowledgebase` class. Figure 5 shows an example implementation of an inference rule. The `rule` object consists of a prerequisite and a fact. The prerequisite evaluates to true or false in dependence of complex terms of the query to the Persistence subsystem. The computed logical value is assigned to the fact.

```
Rule rule = lfs.rule(
  lfs.prereq(
    IntArithmetic.LESS_THAN_OR_EQUAL,
    lfs.cplx(
      fAvgValueDateLocation,
      lfs.variable("sensor type"),
      lfs.variable("date"),
      lfs.variable("location")
    ),
    lfs.cplx(
      fMaxValueDateLocation,
      lfs.variable("sensor type"),
      lfs.variable("date"),
      lfs.variable("location")
    )
  ),
  lfs.fact(
  itIsCold,
  lfs.variable("sensor type"),
  lfs.variable("date"),
  lfs.variable("location"), isCold
  )
);
```

Figure 5. Example of a SensBution rule structure for a location state query.

The Mandarax rules access the event data with `SQLFunctions`. The `SQLFunctions` access the SensBution MySQL [12] database, which implements the database component of SensBution. All sensor events submitted to SensBution are stored in the database (cf. Figure 6).

```
SQLFunction fAvgValueDateLocation =
  new SQLFunction();
fAvgValueDateLocation.
  setDataSource(dataSource);
fAvgValueDateLocation.setQuery(
  "SELECT CAST(AVG(SensorValue)
    UNSIGNED) " +
  "FROM MandatoryTable " +
  "WHERE SensorType= ? " +
  "AND OccurrenceDate= ? " +
  "AND Location = ? ";
  );
fAvgValueDateLocation.
  setObjectRelationalMapping(
    new OneColumnMapping(Integer.class));
fAvgValueDateLocation.setName(
  "avg sensor value");
fAvgValueDateLocation.
  setStructure(funcStructThree);
fAvgValueDateLocation.
  setCloseConnection(false);
```

Figure 6. Example of an SQLFunction for retrieving a sensor value.

RBQuery and RBAnswer Data Transfer Objects

In order to encapsulate the query and answer message information the `RBQuery` and `RBAnswer` are created as custom object classes. They allow to incorporate any information needed to process the content-based queries submitted by the service requester.

The `RBQuery` class implements the query container. It stores the query data as strings. The identity `id` of the query is made up of the object hash, the query data, the peer name, the IP address, and the date to ensure the identification of the query in the whole network. The `dueDate` of an `RBQuery` is of type date. `RBQuery` provides constructors that allow the creation of queries with up to four query variables and the specification of a custom `dueDate`. If the `dueDate` is not specified, a default value is set, so that the peer-to-peer network has enough time to process the query. The `RBQuery` class implements the comparable interface to allow to compare different queries depending on their `dueDate`. This is required to be able to keep the `RBQuery` objects sorted in the ascending order of their `dueDate` in the `PriorityQueue`. The `complexMarker` allows the specification of complex queries that encapsulate up to three simple subqueries.

The `RBAnswer` class implements the answer container. It stores answer data in a `Vector` of `String`. This allows to store the answers of different SensBution peers in one object; the answer of the whole environment can be stored in one `RBAnswer`. `RBAnswer` has a unique identification `id`. It is similar to the identification of the `RBQuery` object that generated the `RBAnswer`. Thus the `RBAnswer` can be referenced by the identification of the `RBQuery`. In order to merge RBAnswer objects they provide a concat method that returns an RBAnswer object encapsulating the answers of two RBAnswer objects with an equal id.

For sending objects of the RBQuery and RBAnswer class over the network, they both implement the Serializable interface.

4.2 Peer Components

In this section we provide details on the implementation of the peers; details on the implementation of other components that are already part of SensBase can be found in [8].

The GatewayXMLRPC class provides its services of the SensBution infrastructure via XML-RPC [1]. It provides an XML-RPC server and implements methods that provide the services of the infrastructure to service requesters. The submitQuery method offered by SensBution allows clients to start a query process, in which they send the query variables as parameters. In order to answer SensBution connects to the deliverAnswer method of the client. The GatewayXMLRPC class encapsulates the query data submitted by the service requester in RBQuery objects and translates RBAnswer objects to raw data that is transferable over XML-RPC.

The singleton class GatewayJXTA implements the Gateway P2P subsystem of SensBution, the central instance responsible for peer-to-peer communication.

At start-up the infrastructure instantiates the GatewayJXTA class, boots the JXTA peer and starts the main thread of the GatewayJXTA class. The main thread receives incoming messages from other SensBution peers and forwards them to dedicated methods that further process the different kinds of messages.

The method responsible for queries messages extracts the RBQuery object and the sender of the query from the query message. It forwards the query object to the QueryHandler for evaluation. It creates an answer message containing the answer object received from the QueryHandler and sends it to the sender of the query.

The method responsible for the answer messages extracts the RBAnswer object from the answer message and stores it in a hashtable with the id of the RBAnswer object as key. New RBAnswer objects and already existing RBAnswer objects in the hashtable with the same id are merged in the order of their arrival. The getAnswers function provides access to the answers in the hashtable. Called with the id of an RBQuery as parameter, it returns the corresponding RBAnswer object and deletes it from the hashtable.

The distributeQuery method takes an RBQuery object as parameter. It encapsulates the RBQuery object in a query message and broadcasts the message to the actual peer group. The GatewayJXTA does not process objects, whose due dates have passed.

The SensBution peer name and the peer group information are stored in the main configuration file of the SensBution infrastructure, to allow for configuration of the GatewayJXTA by the user.

5 Related Work

In the following we will give a brief overview of systems and concepts related to SensBution.

Khronika [11] is a client-server system for event browsing and notification. It was developed to provide users with selected event information. Khronika clients capture events and send them to the Khronika server with event information. The server stores the information in a database. User can search the database and retrieve the manually selected event information. Users receive automatic event notifications by specifying templates of interest. Event daemons compare the templates against new events and send the event information to the receiving clients in case of a match. Khronika is implemented in C and runs on SunOS. The communication between the components is realised using ONC RPC (Open Network Computing Remote Procedure Call).

Elvin [6] is a content-based notification service for collaborative awareness realised as client-server middleware. Clients, so called Producers, detect events and send corresponding event descriptions to a central notification service, the server. Users can specify notification rules. The notification service distributes the event descriptions to the users clients, the consumers, according to these rules. Elvin does not store any event data and hence does not provide a history mechanism. It is implemented in the C programming language and provides client-side libraries for common programming languages.

The *Ubiquitous Service-Oriented Network* (*USON*) framework [15] is a peer-to-peer network architecture for service provision. All entities in the network are self-advertising service elements—that is, they broadcast their own advertisements. Sensor service elements provide basic event information. Service templates combine several different service elements to provide new services. They are user-definable meta-designs for services and may be reused with different instances of the same service elements. USON is implemented on the Java platform using the JXTA protocols [7].

Gaia [13] is a client-server infrastructure for ubiquitous environments where agents provide services: provider agents make available data sources, synthesiser provide inferencing mechanisms, such as rule-engines, a history services store past context information, and consumers that adapt their behaviour depending on the information of providers and synthesisers. Gaia is implemented using CORBA (Common Object Request Broker Architecture).

A task-oriented ubiquitous environment is realised in the *Extrovert Gadgets* project [9]. Tangible objects equipped with sensing, acting, processing, and communication abilities form a wireless ad-hoc network. The objects operate completely autonomous as peers in the network exchanging XML messages. Past events are stored locally in the objects. The system is implemented on the Java 2 Platform, Micro Edition, Connected Device Configuration.

In [10] an information management architecture for fast-evolving ad-hoc networks is introduced, that is not dedicated to ubiquitous environments. It consists of peers with active databases for data storage. Users may submit queries to a peer. The peer creates corresponding ECA (Event-Condition-Action) rules, which act as agents in the ad-hoc network: they are evaluated by the active database of this and other peers in the network. The architecture has not been implemented.

The systems above have the following similarities with SensBution: Khronika and ELVIN are event-based client-server implementations with elementary integration of derivation rules; USON is an event-based peer-to-peer solution; Gaia is a client-server infrastructure for ubiquitous environments with enhanced inferencing and rule support; Extrovert Gadgets is a message-based peer-to-peer environment; and the information management architecture is a concept for rule-agent-based information query in peer-to-peer networks.

6 Conclusions

In this paper we have presented the SensBution concept and implementation providing an integrated event-based and rule-based approach, and supporting client-server and peer-to-peer structures.

For the future several extensions of SensBution are possible such as for instance, the automatic integration of complex replies, and subscriptions to peers.

SensBution peers automatically integrate the low-level results stemming from their own RuleInferencing and from the RuleInferencing of the queried other remote peers. Automatic integrations of results have to be programmed manually (e.g., we explicitly programmed the rules for the room status query). For the future SensBution could also integrate more complex results through the introduction of taxonomies or ontologies.

SensBution instances can publish their services and clients can subscribe to the services. Additional publish-subscribe mechanisms among peers would allow for automatic notifications of sensor event data without the need for a centralised directory service.

Acknowledgments

Thanks to Tareg Egla, Mirko Fetter, and the Cooperative Media Lab (CML) students for contributing to the concepts and implementation of SensBution; and the anonymous reviewers for valuable comments.

References

1. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S. and Winer, D. Simple Object Access Protocol (SOAP). W3C, http://www.w3.org/TR/SOAP/, 2003. (Accessed 15/3/2007).
2. CollabNet Inc. JAL: JXTA Abstraction Layer. CollabNet Inc., http://ezel.jxta.org/jal.html, 2006. (Accessed 7/3/2007).
3. CollabNet Inc. JXTA. CollabNet Inc., http://www.jxta.org/, 2006. (Accessed 15/3/2007).
4. Colmerauer, A. and Roussel, P. The Birth of Prolog. In Second ACM SIGPLAN Conference on History of Programming Languages - HOPL'92 (Apr. 20–23, Cambridge, MA). ACM Press, New York, NY, 1992. pp. 37–52.
5. Dietrich, J. The Mandarax Project. Open Source Technology Group, http://mandarax.sourceforge.net/, 2004. (Accessed 7/3/2007).
6. Fitzpatrick, G., Mansfield, T., Kaplan, S., Arnold, D., Phelps, T. and Segall, B. Augmenting the Workaday World with Elvin. In Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work - ECSCW'99 (Sept. 12-16, Copenhagen, Denmark). Kluwer Academic Publishers, Dortrecht, NL, 1999. pp. 431-450.
7. Gong, L. JXTA: A Network Programming Environment. IEEE Internet Computing 5, 3 (May/June 2001). pp. 88–95.
8. Gross, T., Egla, T. and Marquardt, N. Sens-ation: A Service-Oriented Platform for Developing Sensor-Based Infrastructures. International Journal of Internet Protocol Technology (IJIPT) 1, 3 (2006). pp. 159-167.
9. Kameas, A., Mavrommati, I., Ringas, D. and Wason, P. eComp: An Architecture that Supports P2P Networking Among Ubiquitous Computing Devices. In Second IEEE International Conference on Peer-to-Peer Computing - P2P 2002 (Sept. 5–7, Linköping, Sweden). IEEE Computer Society Press, Los Alamitos, CA, 2002. pp. 57–64.
10. Kantere, V. and Tsois, A. Using ECA Rules to Implement Mobile Query Agents for Fast-Evolving Pure P2P Database Systems. In Proceedings of the 6th International Conference on Mobile Data Management - MDM 2005 (May 9-13, Ayia Napa, Cyprus). ACM Press, New York, NY, 2005. pp. 164–172.
11. Loevstrand, L. Being Selectively Aware with the Khronika System. In Proceedings of the Second European Conference on Computer-Supported Cooperative Work - ECSCW'91 (Sept. 24-27, Amsterdam, NL). Kluwer Academic Publishers, Dortrecht, NL, 1991. pp. 265-278.
12. MySQL. MySQL: The World's Most Popular Open Source Database. http://www.mysql.com/, 2006. (Accessed 15/3/2007).
13. Ranganathan, A. and Campbell, R.H. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In ACM/IFIP/USENIX International Middleware Conference 2003 (June 16–20, Rio de Janeiro, Brazil). Springer-Verlag, Berlin, Germany, 2003. pp. 143–161.
14. Sun Microsystems, I. J2SE 5.0. http://java.sun.com/j2se/1.5.0/, 2007. (Accessed 15/3/2007).
15. Takemoto, M., Oh-ishi, T., Iwata, T., Yamato, Y., Tanaka, Y., Shinno, K., Tokumoto, S. and Shimamoto, N. A Service-Composition and Service-Emergence Framework for Ubiquitous-Computing Environments. In Symposium on Applications and the Internet Workshops - SAINT 2004 Workshops (Jan. 26-30, Tokyo, Japan). IEEE Computer Society, Los Alamitos, CA, 2004. pp. 313–318.