Netquest: An Abstract Model for Pervasive Applications

Version of January 18, 2009

Michel Bauderon[1], Christophe Bobineau[2], Stéphane Grumbach[3], Antoine Henry[3], Xin Qi[3], Wenwu Qu[4], Kun Suo[3], Fang Wang[4], and Zhilin Wu[3]

[1] University of Bordeaux & CNRS-LIAMA
[2] Grenoble Institute of Informatics, Grenoble University, France
[3] INRIA-LIAMA
[4] ISCAS

Abstract. One of the barriers to the implementation and deployment of pervasive applications is the lack of a programming abstraction for ubiquitous computing. We propose a system (i) that supports a declarative programming language, Netlog, well suited to rapid prototyping of distributed applications, (ii) which relies on an embedded DBMS, which facilitates its implementation and its portability over heterogeneous devices, and (iii) supports important functionalities such as implicit routing or adaptive behavior.

1 Introduction

Programming pervasive applications that evolve in an environment distributed over heterogeneous devices raises challenging difficulties. One of them is due to the lack of programming abstraction, that would allow to program an application by specifying globally what the application should achieve, without having to handle details on the architecture of the devices or on the communication networks.

The use of declarative languages has been first proposed in the context of sensor networks to query and aggregate data produced by sensor devices [2]. Queries can be expressed in some restricted SQL, but their execution plan is computed exploiting a global view of the sensor network in a centralized manner on a PC, so not suitable for pervasive applications. The use of declarative rule languages has also been proposed to deal with overlay networks [1] and express network protocols.

Declarative rule languages offer a powerful mean to program pervasive applications. We have designed such a language, Netlog, which extends classical techniques developed in the field of deductive databases [4] to handle the communication and the distribution of the computation. It enjoys a semantics which is mathematically founded, and allows more reliable and secure programs.

The Netquest system relies on the following abstraction: each node perceives the rest of the network as a distributed database and interacts with it by queries.

The query language can be seen as a middleware connecting the ubiquitous components. The novelty of Netquest relies on its implementation on top of a DBMS. Netquest thus runs on devices which are powerful enough to support an embedded DBMS. This offers considerable advantages. First, it can be deployed easily on heterogeneous architectures and networks provided the terminals are equipped with a DBMS. Second, it makes full use of the capacity of the DBMS, including its query optimization and transaction management, which do not need to be reprogrammed for each individual application, resulting in a economy in software development and maintenance.

Finally, Netquest supports easily, due to its high level abstraction, functionalities that are crucial for pervasive applications, such as implicit routing and adaptive behavior.

2 The Netquest System

Netquest supports a declarative rule language, Netlog, which we present through some intuitive examples. The following rules for instance define routes by **recursion**, in a classical way.

$$route(self, dest, dest) :- link(self, dest).$$

$$route(self, neig, dest) :- link(self, neig); route(neig, nh, dest).$$

In this example, *link* represents a direct connection between two nodes, and *route* represents a route between two nodes (first and third parameters) through a third one (next hop, second parameter).

Netlog has the following constructs which are useful to express the dynamic of distributed applications:

- Consumption operator, !, consumes the facts after application of the rule, meaning that it becomes false after being used (deleted from the local database).

  $$helloMAC(metaSelfAddr, metaSelfMac) :- !timeEvent('hello').$$

  This rule generates a *helloMAC* fact. After applying this rule, the timeEvent fact will be deleted.
- Local negation, ¬, is a closed world negation restricted to the local environment.

  $$link(addr, mac, expTime) :- !\ helloMac(addr, mac); \neg\ link(addr, mac, C);$$
  $$expTime := metaSysTime + metaNVP.$$

  The literal ¬ *link(addr,mac,C)* is true if the fact *link(addr, mac,C)* is flase (not in the local database).
- Nondeterministic operator, ⋄, chooses an arbitrary value among a set of possible choices. The following rule chooses one route to dest2 from several route facts with different next hop (*nh*) value.

  $$route(self, \diamond nh, dest2) :- route(self, nh, dest1); link(dest1, dest2).$$
- Communication operators, ↑ and ↓, specify what to do with the facts that are deduced by a rule : ↑ for broadcasting, and ↓ for local storing.

  $$\downarrow route(SelfAddr, SelfAddr, 0, 0, InfPeriod) :- !\ timeEvent('ini').$$

  $$\uparrow helloMAC(metaSelfAddr, metaSelfMac) :- !\ timeEvent('hello').$$

  A *route* fact is deduced and stored locally by applying the first rule, while the *hello* fact deduced by the second rule is broadcasted.

The semantics is defined by iterating the rules till no new fact can be derived. Classical communication protocols (e.g. DSDV, OLSR, DSR) for instance can be written with a few dozen of rules.

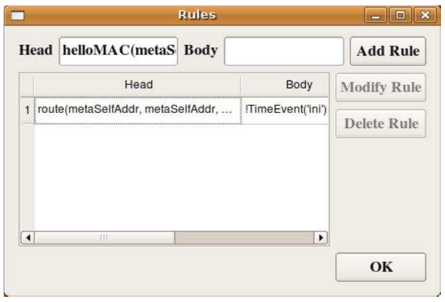
Netlog programs are interpreted by the Netquest system which is to be embedded on all components of the ubiquitous environment. It is implemented on top of an embedded DBMS and a MAC communication layer. It is comprises two main components: a **Distributed Query Engine (DQE)**, and a **Routing Module**. The DQE is the main component of Netquest. It processes queries, generates subqueries to be sent to other nodes and aggregates partial results from other nodes. The Routing Module is in charge of the communication with other nodes, but unlike classical routers, it makes use of the DQE to find routes when needed.

3 Demonstration Features

Netquest is running on top of a network simulation tool, WSNet [3], which allows to simulate the deployment of pervasive applications written in Netlog over a distributed environment. The following components will be used during the demonstration

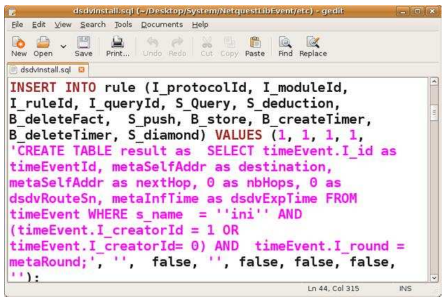
Application Design Environment

The environment, based on GUI, helps the user to design its application in Netlog. The specification and the rules are entered one by one, while the system performs static checks on the Netlog Program. Then the Netlog program is translated into SQL queries. This is a fundamental aspect of Netquest. These SQL queries will be called by the DQE, and executed by the DBMS. In the present prototype, it relies on MySQL. Most of the computation is thus performed by the DBMS, enjoying the local optimization as well as the transaction management of the DBMS. As a Netlog rule can be translated as a set of SQL queries that are to be executed in an atomic manner, rule execution is done using transactions. This principles allows the Netquest system to synchronize concurrent rules execution while guaranteeing the local database coherency. Fig.1 shows the GUI of the Design Environment and Fig.2 shows the SQL queries produced by the compilation.



	Head	Body
1	route(metaSelfAddr, metaSelfAddr, ...	!TimeEvent('ini')

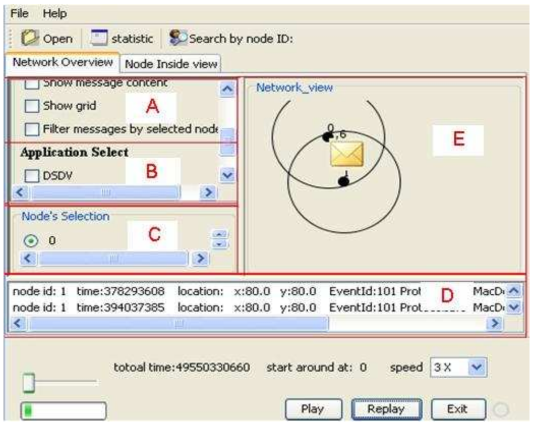
Fig. 1. Design Environment


dsdvInstall.sql (~/Desktop/System/NetquestLibEvent/etc) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
dsdvInstall.sql
Ln 44, Col 315
INS

```
INSERT INTO rule (I_protocolId, I_moduleId,
I_ruleId, I_queryId, S_Query, S_deduction,
B_deleteFact,  S_push, B_store, B_createTimer,
B_deleteTimer, S_diamond) VALUES (1, 1, 1, 1,
'CREATE TABLE result as  SELECT timeEvent.I_id as
timeEventId, metaSelfAddr as destination,
metaSelfAddr as nextHop, 0 as nbHops, 0 as
dsdvRouteSn, metaInfTime as dsdvExpTime FROM
timeEvent WHERE s_name  = ''ini'' AND
(timeEvent.I_creatorId = 1 OR
timeEvent.I_creatorId= 0) AND  timeEvent.I_round =
metaRound;', '',  false, '', false, false, false,
'');
```

Fig. 2. Corresponding SQL Queries

Simulation of the network First, the parameters of the network to be simulated are specified, including MAC protocol, radio range and battery consumption. WSNet, the event-driven simulator for wireless networks, then simulates a topology with the first and the second communication layers.
Simulation of the application While running the simulation, the Netquest system produces logs which are used to visualize the execution. Fig.3 shows the visualization tool.



```
node id: 1   time:378293608   location:  x:80.0  y:80.0   EventId:101 Prot        MacD
node id: 1   time:394037385   location:  x:80.0  y:80.0   EventId:101 Prot.......  MacD
```

totoal time:49550330660   start around at: 0   speed 3 X

Zone A selected content to be displayed, such as node id, message id, radio range, etc;
Zone B applications or protocols selection;
Zone C nodes whose messages exchange is to be displayed;
Zone D node's events during simulation;
Zone E network with moving nodes, message exchanges, which can be click on to display their content.

Fig. 3. Visualization Canvas

The local database of each node contains the data related to both applications and network management. Both are handled by queries in Netlog. On the simulation, all these aspects can be visualized, i.e. messages exchanged between nodes and local database of each node, which can be monitored dynamically. A consistent part of the computation is performed by the DBMS, which executes the SQL queries on calls from the DQE. The router routes further queries to other nodes if the destination is explicit and known using the DQE and local DBMS if needed. The Netlog language increases the security and the reliability of the applications thanks to its well defined semantics.

The Netquest system supports interesting functionalities such as implicit routing which can be easily expressed as Netlog queries, as well as adaptive behavior, with rules adapting to the network environment, and as a result of distributed query optimization done by DQE.

References

1. Boon Thau Loo et al. Implementing declarative overlays. *SOSP*, 2005.
2. David Chu et al. The design and implementation of a declarative sensor network system. In *SenSys '07*. ACM, 2007.
3. Guillaume Chelius et al. Worldsens: a fast and accurate development framework for sensor network applications. In *SAC '07*, http://wsnet.gforge.inria.fr/.
4. Raghu Ramakrishnan and Jeffrey D. Ullman. A survey of research on deductive database systems. *Journal of Logic Programming*, 23:125–149, 1993.

Demo Sketch

The demonstration goes along the three following phases:

Application design The application is programmed in Netlog using the programming environment, the program is verified and translated automatically into SQL queries;

Network simulation A network is randomly generated with the desired parameters using WSNet;

Application simulation Visualization of the execution of the application, with the exchange of messages, and the updates of local databases of each node.

Application design
We will first show Netlog programs that have been written to express various problems including communication protocols (e.g. DSDV, OLSR, DSR, FISCO, etc.), and some distributed applications. The objective is to illustrate the potential of the Netlog language to express complex functionalities such as adaptive behavior.

In a second step, the user will be invited to design its own application, either from scratch, or by modifying already written Netlog programs.

Network simulation
The user is required to choose the parameters of the network configuration to generate the network on which the demo will be ran. The assumptions taken might depend upon the targeted applications.

Application simulation
Combining the two previous steps, we will simulate the execution of the application written in Netlog on the generated network. Several objectives will be pursued in the simulation. First, show that the declarative specification does not slow down the execution. We will provide comparisons of classical implementations of protocols, with the execution of the corresponding Netlog program, in terms of number of messages, time, etc. Second, we will illustrate the use of the database, which simplifies greatly the code, both of the Netquest system itself, but also of the Netlog programs.