Using the Traceability Data Generator

Intelligent Information Systems
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

April 12, 2006

This document describes how to use the Traceability Data Generator developed at the IBM Almaden Research Center. The data generator can be used to produce event data for a traceability networks. Please contact the authors for further information on traceability networks and architectures for systems used for the data generator.

1 Requirements

Apart from the data generator executable, the following software has to be installed:

- Java virtual machine, version 1.4.2 or higher
- Derby database runtime, version 10.1.2.1 or higher

2 Installation

The data generator requires no separate installation. However, it is advised to create an empty directory folder and store the data generator executable as well as the Derby database runtime in it. The data generator is creating a Derby database instance when started for the first time. This instance will be created in the current directory at runtime.

3 The Data Generation Process

The data generator produces event and entity data for three possible architectures: a data warehouse architecture, a distributed architecture with

central object naming and discovery services (EPCglobal approach) and for process-and-forward.

After an entity is created at a randomly chosen node (cf. 4.1), one of four actions is taken for that entity. It is either shipped to a neighboring node, stays at the current node for one cycle, is associated with an existing entity, or it is dropped by the current node. The action is chosen randomly in accordance with the provided probabilities. For the process-and-forward approach appropriate entries for *receivedFrom* and *sentTo*, and for the EPC-global approach data entries for the Discovery Service are created.

This process is repeated until either the entity is dropped or the path of the entity cannot be extended further. Afterwards, the process starts again for the next entity until data for the desired number of entities has been generated.

3.1 Output Format

In the following, the relational schema for the generated data is presented. The generated data is stored in in one file for each relation and each node. The tuples are stored in the form of comma separated values for easy uploading into a databases.

Four relations are universal in the sense that they are part of the data model and therefore present in all three architectures (data warehouse, EPC-global, process-and-forward). In the two distributed architectures, these tables are present at each node in the network and one file for each of those tables for each node will be created.

The relation *event* contains all sensing events and the relation *location* some address information, *i.e.* some description, of the locations under the control of a node. the relation *s_prop* stores static data about entities, *i.e.* attributes that do not change over the lifetime of the entity. The attributes for which data is generated are the entity's color (a string randomly chosen from a set of configurable values; see 4.1), and the minimal and maximal allowed storage temperature for this entity (integer value randomly chosen from a range of -100 to +100).

The contains relation stores which entities are associated with each other, that means which objects are put into other objects at a particular location and point in time. For all locations and time-points thereafter, only events for the outer, containing entity are generated, but not for the contained entities anymore.

event*(eid char(12) not null, lid varchar(50) not null,*
ts timestamp not null, primary key (eid, lid, ts))

location*(lid varchar(50) not null primary key,*
address varchar(100))
s_prop*(eid char(12) not null primary key, color varchar(20),*
minTemperature Integer, maxTemperature Integer)
contains*(eid char(12) not null, contained_eid char(12) not null,*
ts timestamp not null, primary key (eid, contained_eid, ts))

If data for the EPCglobal approach is generated, two relations for the central services Object Naming Service (ONS) and Discovery Service (DS) are created. While the ONS relation stores at which location an entity was initially created, the discovery service keeps track of which nodes in the network have seen that entity during its lifetime.
ONS*(eid char(12) not null, dbid varchar(50) not null,*
ts timestamp not null, primary key(eid, dbid))
DS*(eid char(12) not null, dbid varchar(50) not null,*
ts timestamp not null, primary key(eid, dbid, ts))

For the process-and-forward approach information about the route an entity took is not stored in a central service but instead is distributed over the *recievedFrom* and *sentTo* relations present at each node. The data generated for those relations adheres to the following schemata:
receivedFrom*(eid char(12) not null, dbid varchar(50),*
ts timestamp not null, primary key(eid, ts))
sentTo*(eid char(12) not null, dbid varchar(50),*
ts timestamp not null, primary key(eid, ts))

3.2 Starting the Data Generator

To start the data generator, the Derby library has to be in the classpath. The following command starts the data generator on a UNIX/Linux platform with configuration file *generator.cfg*:

```
java -classpath DataGenerator.jar:db-derby-10.1.2.1-lib.zip ↩
com.ibm.traceability.dataGenerator.Generator generator.cfg
```

4 Parameter Settings

The configuration of the data generator is stored in two files: the configuration file and the network topology file.

4.1 Configuration File

The configuration file contains all parameters needed for data generation, including the position of the network topology file. It is a text file containing name-value pairs in a Java Properties format. In the following, all possible parameters are explained. For mandatory parameters, a value has to be provided in the configuration file. For optional parameters, the default value is listed. Figure 1 shows an example of a configuration file.

entities *(mandatory)* The number of entities which move through the network and leave traces. This parameter mainly determines the size of the output.

centralized *(optional; default: false)* If this option is set to *true*, data for a central data warehouse is produced.

epcglobal *(optional; default: false)* If this option is set to *true*, data for a distributed approach with central Object Naming and Discovery Service is produced.

processandforward *(optional; default: true)* If this option is set to *true*, data for a distributed process-and-forward architecture is generated.

The options *centralized*, *epcglobal* and *processandforward* can all be set *true* at the same time, thus allowing to generate equivalent event data for all three architectures.

topology *(mandatory)* Sets the name and path to the network topology file.

types *(mandatory)* The entities generated are randomly assigned a type. The type of an entity is a string without any whitespace. The value of the *types* parameter is a list of possible type names separated by comma or whitespace.

startpoints1-n *(optional; default: none)* For each entity type a set of start nodes can be defined. The value is a list of IDs of network nodes (see section 4.2), separated by comma or whitespace. When an object of the corresponding type is created, one of those startpoints is chosen randomly as the node at which the entity was created. The number of the *startpoints* parameter is the position of the corresponding type in the types list, *e.g.* the value of the parameter *startpoints1* contains the allowed startpoints for the first type in the list of types. If no startpoints for a specific type are provided, an arbitrary node from the network is chosen randomly as startpoint.

colors *(mandatory)* List of possible colors for entities. The value is a list of strings without whitespace, separated by comma or whitespace.

extensionProbability *(mandatory)* The probability for the path of an item being extended. This probability is multiplied with *extensionDecrease-*

```
# configuration file for the traceability data generator

# number of entities to create
entities=100

# determin ethe architectures for which a data is produced
centralized=false
ecpglobal=false
processandforward=true

# filename from which the topology is read/is written to
topology=ExampleNetwork.xml
# possible object types
types=car,engine,cylinder,screw
# the IDs of the nodes which may be starting points for entities
startpoints1=EPCIS2
startpoints2=EPCIS1
startpoints3=EPCIS1
startpoints4=EPCIS1
# possible colors for objects
colors=red,green,blue,yellow,white,black

# start probability with which an entity path is extended
extensionProbability=0.6
# the factor used to decreased extensionProbability at each step
extensionDecreaseFactor=1.0
# the probability with which at the end of a generated path an
# assembly step is tried
assemblyProbability=0.33

# after how many generated entities is the master clock advanced
clockAdvance=50
# how many seconds is one clock step
clockStep=3600
# how many seconds is one internal clock step (between two shipments
# within one enterprise)
internalClockStep=60
```

Figure 1: generator.cfg: An example configuration file.

Factor after every path extension step.

extensionDecreaseFactor *(mandatory)* The factor by which the *extensionProbability* is decreased after each extension of an entities path. To turn off this feature set the value to 1.

assemblyProbability *(mandatory)* Probability of an assembly event as opposed to an path extension.

clockAdvance *(mandatory)* Number of entities after which the master clock is advanced one *clockStep*.

clockStep *(mandatory)* Length of time interval (in seconds) that passes when an entity is transfered between two nodes.

internalClockStep *(mandatory)* Length of the time interval (in seconds) between two events taking place at different locations within one node for the same entity.

4.2 Network Topology File

The network topology file describes the nodes and the entity flow between them. The network is described in an XML format. Figure 2 shows an example of a network topology file.

A network topology file contains exactly one *EPCNetwork*, which consists of a list of *node* elements followed by a list of *edge* elements. Nodes have an *id*, followed by a *failureRate* and a list of *location*. *id* is an arbitrary string uniquely identifying the node within the network. *failureRate* denotes the probability with which an entity remains at this node forever. The *failureRate* has to be a number between 0 and 1.

The list of *location* contains the identifiers of the locations under the control of the organization represented by the node. The number of locations per node must be greater or equal to 1 and the location names must be unique within the network. Apart from those requirements, the location identifiers can be arbitrary strings.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Example network for the traceability data generator -->
<EPCNetwork>
<node>
      <id>EPCIS1</id>
      <failureRate>0.01</failureRate>
      <location>GLN1</location>
      <location>GLN2</location>
</node>
<node>
      <id>EPCIS2</id>
      <failureRate>0.05</failureRate>
      <location>GLN3</location>
</node>
<node>
      <id>EPCIS3</id>
      <failureRate>0.1</failureRate>
      <location>GLN4</location>
</node>
<edge>
      <source>EPCIS1</source>
      <target>EPCIS2</target>
      <weight>0.9</weight>
</edge>
<edge>
      <source>EPCIS2</source>
      <target>EPCIS3</target>
      <weight>1.0</weight>
</edge>
<edge>
      <source>EPCIS1</source>
      <target>EPCIS3</target>
      <weight>0.1</weight>
</edge>
</EPCNetwork>
```

Figure 2: EaxmpleNetwork.xml: An example network topology file.