Introduction to Composer

Carlos Castro Irgorri

Business Network Application

In Composer the BNA (*.bna) is made of:

- Model file (*.cto)
- Script file (*.js)
- Query File (*.qry)
- Access Control File (*.acl)
- README.md

Business Network Application

In Composer the BNA (*.bna) is made of:

- Model file (*.cto)
- Script file (*.js)
- Query File (*.qry)
- Access Control File (*.acl)
- README.md

Use Yeoman generator to build an empty of tutorial bna:

> yo hyperledger-composer

Business Network Application

A simple object oriented language .

In the business domain model we define the resources:

- Participants
- Assets
- Transactions
- Events

Model File (namespace.cto)

1. Namespace (unique name): resource declaration on this file.
2. Define resources: assets, participants, transactions, events.
3. It is possible to import resources, that is the business domain model can be spread over different files (*.cto).

```
import org.degree.Administrator
Import org.degree.*
```

System Namespace (org.hyperledger.composer.system) defines the class of basic resources (asset, event, transaction..).

Resource definition (clases)

- Assets y participants must be perfectly identified (**identified by**). Resource definition is similar to defining objects in Javascript:
- Primitive types:
  1. String: a UTF8 encoded String.
  2. Double: a double precision 64 bit numeric value.
  3. Integer: a 32 bit signed whole number.
  4. Long: a 64 bit signed whole number.
  5. DateTime: an ISO-8601 compatible time instance, with optional time zone and UTZ offset.
  6. Boolean: a Boolean value, either true or false.

Assets and Participants

Characterized by a list of fields:

participant Administrator identified by email{
- String email
- String name

}

asset Certificate identified by CertId {
- String CertId
- String Program
- Langue idiomac

}

Enumeration

Enumeration: a list of predefined values:

enum Langue {

- German
- Italian
- French

}

Arrays

- String[] contributor

“contributors”:[“juan”,”daniel”,”miguel”]

Arrays can extend primitive and other user defined variables.

-- > Langue[] idiomas

Assigning values to different fields

- By definition all fields are required except if we define them as:

optional

- Some fields may be instantiated with a particular default value:

default = “French”.

Elements from object orientation

- Abstraction/abstract: define a ”generic” resource as abstract (it can not be created individually) it must be extended by a resource that is not abstract.
- Inheritance/extends: resources that burrow properties and fields (of the same class) and include other fields.
- Association/concept: abstract class. It is not a resource (no identity required). A way to group field that are related to each other. It can be abstract and extended.

Field validation

Field validation uses regular expressions:

- regex=

E-mail: ^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$

Test: regex101.com

- range=

[100,] 100=> o [,100] <=100 o [5,10]

Registry

- The information associated to each asset or participant is kept in a registry.
- In the registry we have a unique identifier for each resource, this id corresponds to the field marked as *identified by*.

Relationship operator

- A resource can reference types that are defined on other resources.
- These types of relationships are in only one direction
- *Pointer to a specific asset type*. (asset to participant)

```
asset Certificate identified by  CertId {
  String CertId
  String Program
-- > Administrador  administrador
}
```

Using resources

A resource can be identified, related and assigned using the following elements::

namspace.resourceName#identifier

For example:

org.degree.Administrator#casaur@urosario.edu.co

Relationships are one-directional and static, that is if we eliminate a resource the relationship is maintained.

Transactions (logic.js)

Transactions are the actions that participants execute over assets. These actions are reflected in the ledger (log file y state DB).

transactions have two identifiers:

- transactionId
- timestamp

These identifiers are assigned automatically by the ledger (Fabric)

JavaScript (composer), Go (Fabric)

Transactions (logic.js)

We write the transaction logic in JavaScript and its relationship to the model file (*.cto):

```
/* input */
@param {org.degree.RecordDegree}  data
@transaction
function RecordDegree(data){
/*operation: assign new information to a field*/
data.asset.attributeName=data.sourceName;
```

Transactions (logic.js)

Identify the type of registry that is affected by the transaction asset or participant:

```
/*function that allows to access the registry*/
    getAssetRegistry('namespace.Asset')
```

Update the registry with the new information

```
    assetRegistry.update('namespace.Asset')
}
```

Types of registry

- AssetRegistry
- ParticipantRegistry
- TransactionRegistry
- Historian: only read the registry
- IdentityRegistry: only read the identities.

The function getTipyofRegistry(RegistryName),

Allows access to the elements of the registry. getAll.

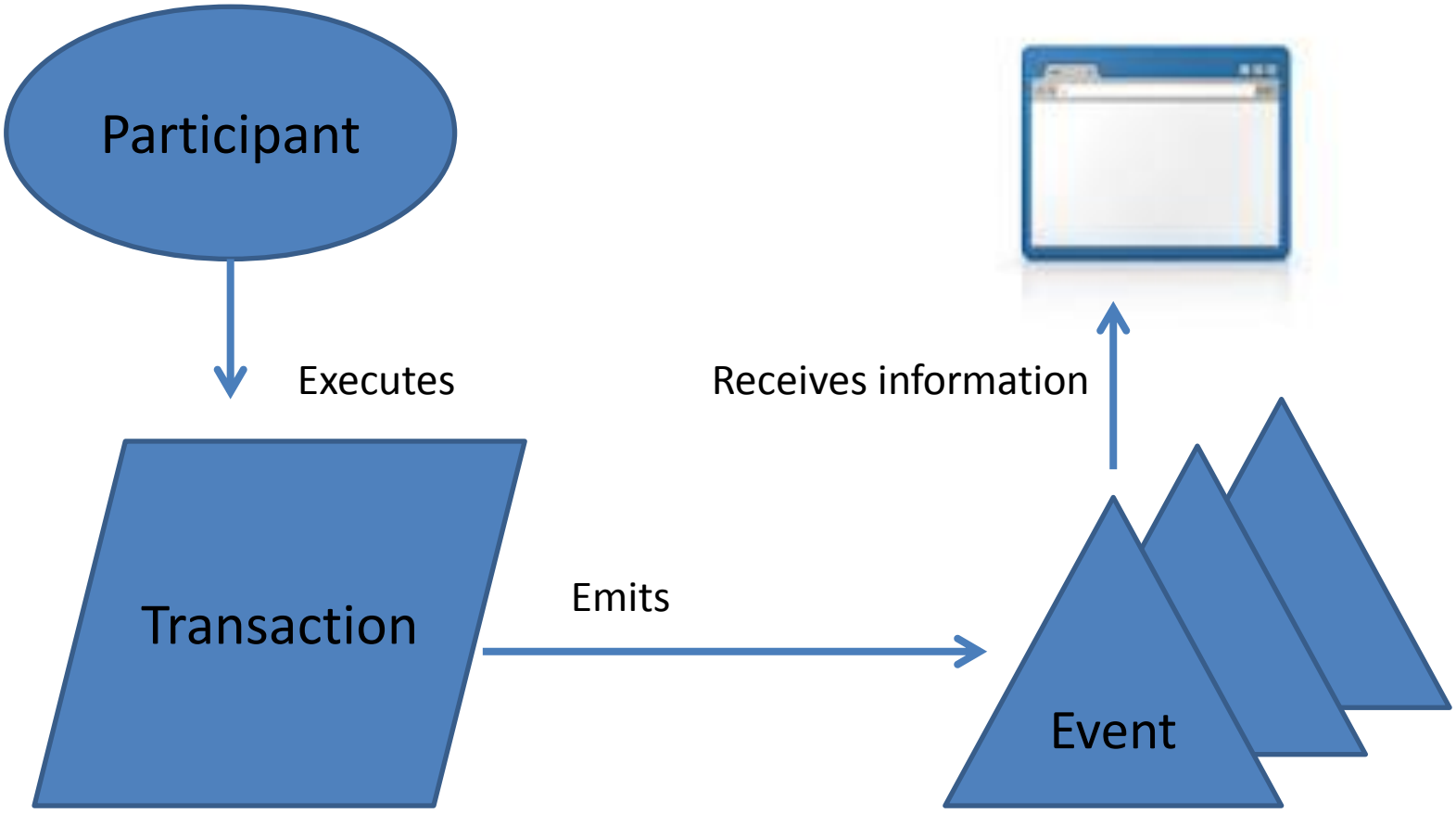
Classes associated to the registry

Actions on the resources that are executed over the registry:

- Add
- Update
- Remove
- Get
- Check
- Resolve

Hyperledger Composer API

Events


Participant
Executes
Transaction
Emits
Receives information
Event

Events

Similar to transactions, events have two identifiers:

- transactionId
- timestamp

Events are emitted from the transactions indicating that something has happened in the *ledger.* Different applications may subscribe to the events using *composer-client API*

Events

Emit an event in a transaction that updates the balance in a wallet.

```
let event = getFactory().newEvent('org.bforos', 'WalletEvent');
    event.claimer = claimData.claimer;
    event.oldbalance = balance;
    event.newbalance = balance+points;
emit(event);
```

getFactory() function provides access to the resources defined in the bna.

Factory Class Composer

Historian

Registry of all successful transactions:

- Asset defined by the system
- Transactions defined by the system.
- It is possible to launch queries (queries) over the Historian registry.
- Historian in Playground: All Transactions

Query language

Its possible to search over the resource registry using: SQL like query language:

- Named Query
  - Defined in BNA (*.qry)
  - Use the composer-rest-server
- Dynamic Query
  - Dynamically constructed.
  - Defined at the transaction level or the *Client API*

Named Query, queries.qry

```
query selectHistorianRecordsByTrxId {
description: "Select historian records by
transaction id"
statement:    SELECT
org.hyperledger.composer.system.HistorianRecord
WHERE (transactionId == _$transactionId)
}
```

Statement supports SQL: LIMIT, ORDER BY,…

Access to parameters: _${param-name}

Access control (*.acl)

- If we do not define explicit access control then we have by default access to every resource.
- Management of identities has various levels:
  - Network Administrator, network level.
  - Peer Administrator, node level.
  - Participants, within a bna.

Simple rules for access control (*.acl)

Access control to the **namespace, assets** or ownership of an asset by a **participant**

```
rule Default {
description: "Allow all participants access to all resources"
participant: "ANY"
operation: ALL
resource: "org.bforos.*"
action: ALLOW
}
```

Composer Access Control Language

Business Network Application

After we complete the model we can create the BNA (*.bna) –network archive file- using Playground or Composer CLI (command line interface):

Identify the folder (dir) that contains the files for the BNA

> composer archive create –a degree.bna –sourceType dir -- sourceName

Composer CLT commands