Introducción a Composer

Carlos Castro Irgorri

Business Network Application

En Composer se define el BNA (*.bna)

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

Business Network Application

En Composer se define el BNA (*.bna)

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

Se puede inicializar un esqueleto de modelo con:

> yo hyperledger-composer

Business Network Application

Lenguaje (sencillo) de modelación orientado a objetos .

En el business domain model se definen los recursos:

- Participants
- Assets
- Transactions
- Events

Model File (namespace.cto)

1. Namespace (nombre unico): recursos declarados en el archivo.
2. Definicion de Recursos: assets, participants, transactions, events.
3. Es posible importar recursos, es decir que el business domain model puede estar repartido en varios archivos (*.cto).

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

System Namespace (org.hyperledger.composer.system) define clase de recursos basicos (asset, event, transaction..).

Definición de recursos (clases)

- Assets y participants deben estar perfectamente identificados (**identified by**). La definición de recursos es similar a la definición de objetos en 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.

Activos y Participantes

Caracterizados por una lista de campos:

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

}

asset Certificado identified by CertId {
- String CertId
- String Programa
- Idioma idiomac

}

Enumeracion

Enumeration: lista de valores predefinido por el usuario para una variable:

enum Idioma {

- Aleman
- Italiano
- Frances

}

Arrays

Permitir la definición de campos con un conjunto de valores :

- String[] contributor

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

Se puede extender arrays de primitivos a otras variables definidas por el usuario tambien.

-- > Idioma[] idiomas

Asignación de valores en campos

- Por definición todos los campos son obligatorios salvo que se caractericen como:

optional

- Algunos campos se pueden instanciar con un valor en particular utilizando:

default = “Frances”.

Elementos de orientación de objetos

- Abstraction/abstract: marcar un recurso (generico) como abstracto (no se puede crear por si mismo) se debe extender con un recurso no abstracto.
- Inheretance/extends: recursos que recogen propiedades o campos de otros recursos (del mismo tipo) y agrega otros campos.
- Association/concept: clases abstracta. No es un recurso (no requieren identidad). Forma de agrupar campos con relaciones entre ellas. Puede ser abstracto y extenderse.

Validación de campos

La validacion de campos utiliza Regular expressions de tipo:

- 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]

Registros

- La información asociada a cada asset o participant se mantiene guardada en un registro.
- En el registro se tiene un identificador único para cada recurso que corresponde al campo indicado por *identified by*.

Operador de Relaciones

- Un recurso puede hacer referencia a tipos que están definidos en otro recurso.
- Este tipo de relaciones solo va en una dirección.
- *Pointer to a specific asset type*. (asset to participant)

```
asset Certificado identified by  CertId {
  String CertId
  String Programa
-- > Administrador  administrador
}
```

Relaciones a recursos

Un recurso puede ser identificado, relacionado y asignado a partir de los siguientes elementos:

namspace.nombreRecurso#identificador

Por ejemplo:

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

La relaciones son unidireccionales y estáticas, es decir si un elemento se elimina la relación se mantiene.

Transacciones (logic.js)

Transacciones son las acciones que un participant realizan sobre un asset, implican operaciones sobre el registro (log file y state DB).

transaction tiene dos identificadores:

- transactionId
- timestamp

Asigna automáticamente la red (Fabric)

JavaScript (composer), Go (Fabric)

Transacciones (logic.js)

En JavaScript se escribe la lógica de la transaccion y su relacion con la definicion en model file (*.cto):

```
/* input */
@param {org.degree.RecordDegree}  data
@transaction
function RecordDegree(data){
/*operación: asigna nueva información a un campo*/
data.asset.attributeName=data.sourceName;
```

Transacciones (logic.js)

Identifica el tipo de registro sobre el cual actúa la transaccion: un asset o participant

```
/*funcion que permite acceder a registro*/
getAssetRegistry('namespace.Asset')
```

Actualiza el registro con la nueva información

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

Tipos de registro

- AssetRegistry
- ParticipantRegistry
- TransactionRegistry
- Historian: solo lectura de los registros
- IdentityRegistry: solo lectura de las identidades.

La funcion getTipodeRegistro(RegistryName), permite acceder a elementos del registro.

getAll.

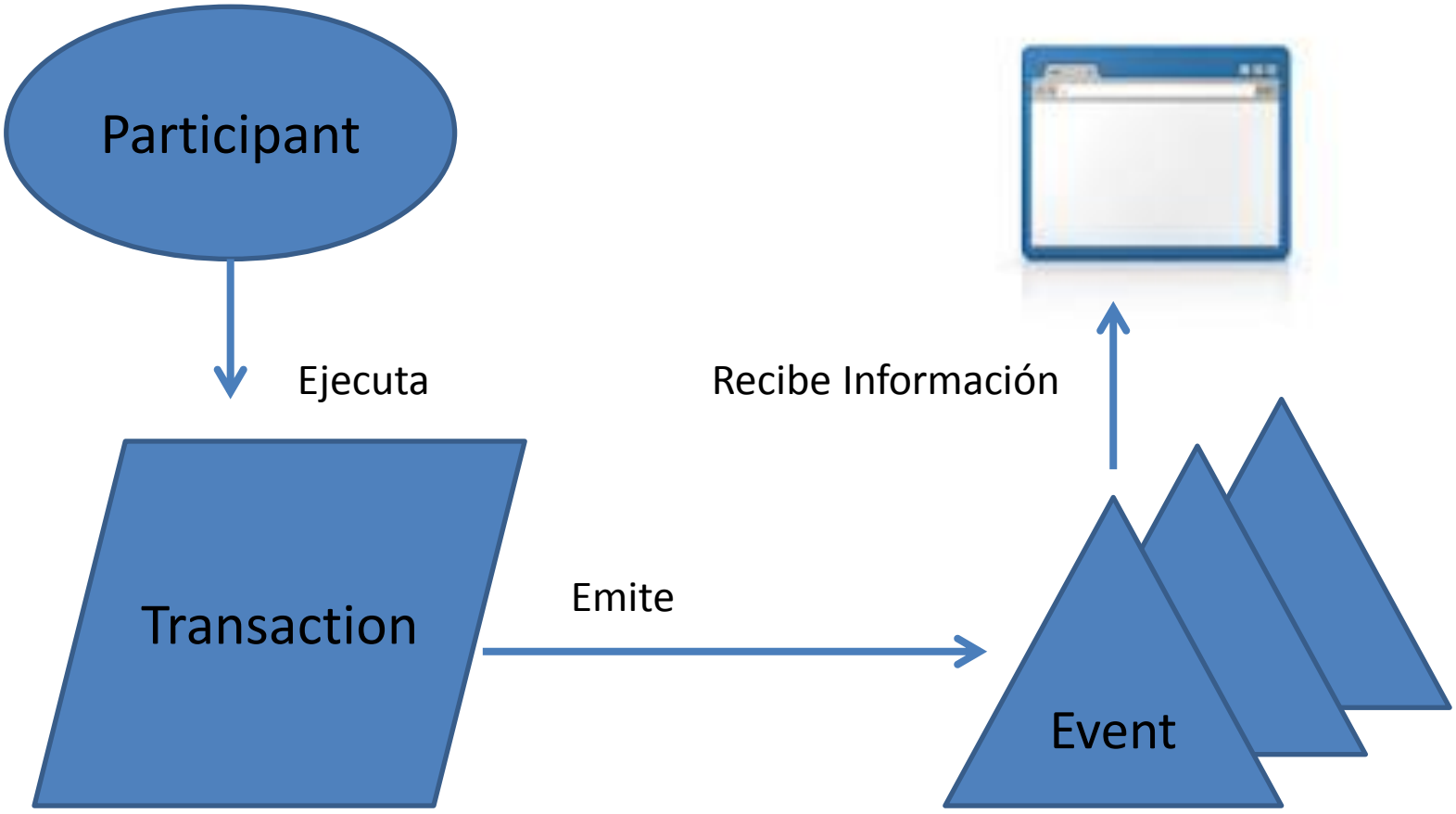
Clases asociadas a los registros

Acciones sobre los recursos que se permiten sobre el registro:

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

Hyperledger Composer API

Eventos


Participant
Ejecuta
Recibe Información
Transaction
Emite
Event

Eventos

Al igual que las transacciones, los eventos tiene dos identificadores:

- transactionId
- timestamp

Los evento se emiten a partir de las transacciones indicando que algo ha sucedido en el *ledger.* Diferentes aplicaciones se pueden subscribir a los eventos utilizando *composer-client API*

Eventos

Emisión de eventos en la transacción que actualiza un balance de una billetera.

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

getFactory() funcion para acceder a instancias de los recursos definidos en el bna.

Factory Class Composer

Historian

Registro de todas las transacciones exitosas:

- Asset definido por el sistema
- Transacciones del sistema
- Es posible lanzar búsquedas (queries) sobre el registro Historian.
- Historian en Playground: All Transactions

Lenguaje de busqueda

Es posible hacer busquedas sobre el registro de recursos utilizando: SQL like query language:

- Named Query
  - Definidas en el BNA (*.qry)
  - Utilizar a través de composer-rest-server
- Dynamic Query
  - Se construyen dinámicamente.
  - Se encuentran dentro de las transacciones o a través del *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 soporta lenguaje SQL: LIMIT, ORDER BY,…

Permite acceder a parametros: _${param-name}

Control de acceso (*.acl)

- Si no se define un control de acceso explicito entonces hay acceso completo a los recursos.
- La administración de identidades tiene varios niveles:
  - Network Administrator, a nivel de la red.
  - Peer Administrator, a nivel del nodo.
  - Participants , dentro de la bna.

Reglas simples de control de acceso (*.acl)

Control de acceso al **namespace, assets** o propiedad de un activo por parte de un **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

Despues de completado el modelo se puede crear el BNA (*.bna) –network archive file- a partir de Playground o Composer CLI (command line interface):

Identificar la carpeta (dir) que contien los archivos de BNA

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

Composer CLT commands