

Blockchains Architecture, Design and Use Cases
Prof. Sandip Chakraborty
Prof. Praveen Jayachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 28
Hyperledger Composer – Application Development

Hello everyone. Welcome back to the next lecture of our Blockchain course. So, you now got the good hands on experience on what fabric is, how to use hyperledger fabric and so on. In the next 2 lectures we are going to see a another very interesting hyperledger project called hyperledger composer. And this is primarily intended to help developers, business users to create blockchain applications in a very simple and intuitive manner. Today with hyperledger fabric you actually need to understand a good deal about how fabric is architected.

(Refer Slide Time: 00:42)

Hyperledger Composer: Accelerating Time to Value

- A suite of high level application abstractions for business networks to be built on top of Hyperledger Fabric
- Emphasis on **business-centric vocabulary** for quick solution creation – model in terms of assets, participants and transactions
- Reduce risk, and increase understanding and flexibility

<https://hyperledger.github.io/composer>

Business Application

↓

Hyperledger Composer

↓

Blockchain
(Hyperledger Fabric)

- Features
 - Model your business networks, test and expose via APIs
 - Applications invoke transactions to interact with business network
 - Integrate existing systems of record
- **Fully open and part of Linux Foundation Hyperledger**
- Try it in your web browser now:
<http://composer-playground.mybluemix.net/>

3

You need To understand ordering service, peers, the certificate authority MSPs, you need to understand identities, you have to work with certificates, you have to have digital signing. That a lot of complexity is there, that only a an experienced developer might be comfortable with. It takes a little while to get a hang of all of these concepts.

Hyperledger composer wants to make that all that process of development a lot easier for you. A development and management of your blockchain network. So, it is really targeted towards business users. It is on the emphasis is on developing a business centric vocabulary,

to define what you are blockchain is going to be. So, you are going to be defining your smart contract in terms of models, you are going to model it in terms of assets. So, instead of thinking a low level variables, data elements you are going to be talking in the higher level business assets. You will be talking in a in the notion of participants, you do not have to work with certificates any more.

And you will be talking about business logic or business transaction. So, one of that is abstracted one level up for you to define your blockchain network. And this we hope will significantly reduce the risk of development, and the fact that it takes long time to develop some of these applications and can be fairly complex. It also increases understandability and flexibility.





So, if when you see a composer code, then you can easily understand what this code is trying to do, and you are not at you not working with a very high low level programming language; like, go lang or java. Now what are some of the features like I mention? It is helps you model business network. And it also helps you test an expose via API's. So, composer has a way to automatically generate certain API's based on the assets that you define.

And applications can invoke transactions to interact with the business network um. And it also provide some support for integrating your existing systems your organizations internal systems with blockchain. This is fully open source it is opened and governed and it is part of the Linus foundation. It is under the hyperledger project itself. It is another project in hyperledger. The current implementation of composer works on top of hyperledger fabric. So, there is a hyperledger fabric network that you have setup. And composer allows you to do application development at a higher level business level compared to what hyperledger fabric allows you to do.

But it is possible that in future composer could connect to other blockchain platforms as well. Today it only connects to hyperledger fabric, that is what it is shown in this picture there is a business application, there is a composer layer and then there is fabric. So, today it connects only to fabric, but in future it could connect to other platforms as well. There is a very easy to use web browser playground. So, you can go to the playground you can start playing around with it, you can get a feel for what application development looks like in the composer world. And the same code you can actually you can get the download the code setup yourself and create a playground just for yourself. So, that is also possible.

(Refer Slide Time: 03:49)

Goals of Hyperledger Composer

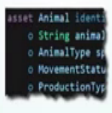


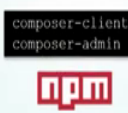

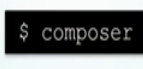


			
Increase understanding	Save time	Reduce risk	Increase flexibility
Bridges simply from business concepts to blockchain	Develop blockchain applications more quickly and cheaply	Well tested, efficient design conforms to best practice	Higher level abstraction makes it easier to iterate

4

So, what are the main goals of composer why did we create this? It is to increase understanding of business applications, you can think of coding your smart contracts from business context. You can hopefully save significant amount of time in the development process itself. It helps you reduce risk, because now your coding is a lot simpler. It is not have to deal with the netegrity details of fabric. And it hopefully also increases your flexibility, and how you abstract your concepts, and it makes it easier to (Refer Time: 04:20) so, you can improve upon your code fairly easily.

(Refer Slide Time: 04:24)

Extensive, Familiar, Open Development Toolset

 Data modelling	 JavaScript business logic	 Web playground	 Client libraries
 Editor support (Atom, Visual Studio)	 CLI utilities	 Code generation	 Existing systems and data

5

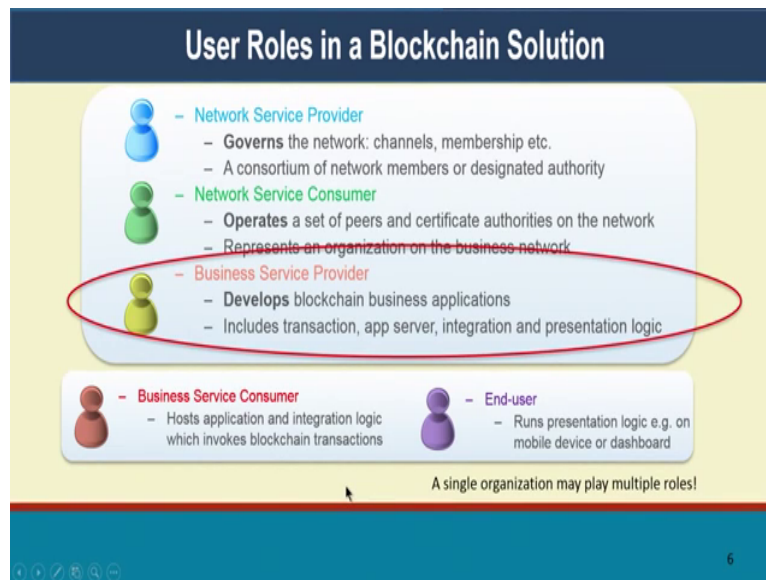
So, composer is has been developed using a fairly extensive open development toolset. This should be very familiar to you if you are a JavaScript fan. So, it is entirely written in JavaScript. So, there is a custom modelling language is again very intuitive to use it model notions of assets participants and transaction we will talk about that. We will talk about some of the business logics that can be written in JavaScript, and that can be run as part of your smart contracts. So, we will talk about that. There is a web playground, that you can play around with then understand how to create applications.

There are couple of libraries; a client libraries, composer hyphen client, composer hyphen admin. That you would use to connect your blockchain application to the fabric network. And these are provided as npm module. For couple of very prominent well highly used editors, there is a support for writing your composer applications on them. So, we support atom and visual studio currently, there could be others in the future CLI utilities are provided. So, you can invoke composer commands directly using CLI.

And there is of course, API's as well. And composer also gives you the ability to generate code. So, this is specifically for people who want to generate quickly generate a user interface for their application. There is yeoman it is a popular tool for code generation. So, based on how you have defined your assets and other modelled other things in your network. You can automatically generate code that can be connected to a user interface.

And it also follows existing standards with loopback. And swagger to connect your existing systems and access the data through that. So, it provides an easy way for you to generate a set of API's for your specifically for your application. And then you can use your API's in other applications.

(Refer Slide Time: 06:21)



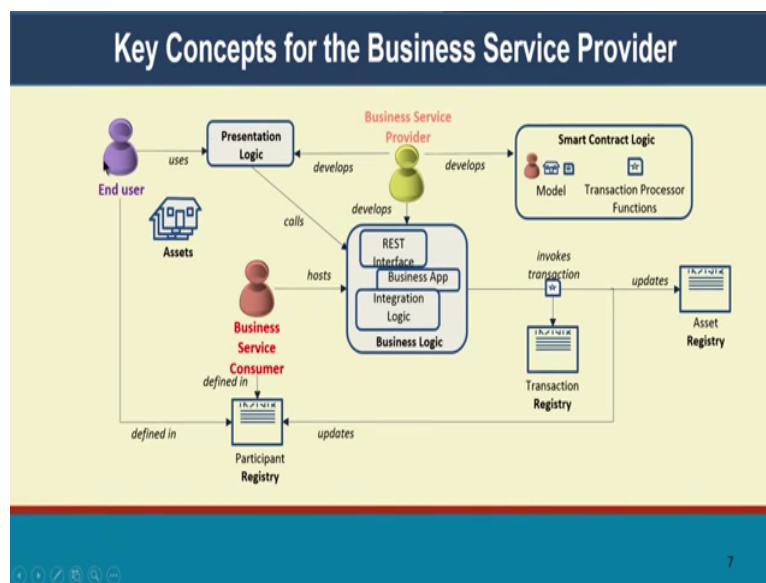
So, we will look at some these concepts in a little more detail. So, the specific there are multiple user roles in a blockchain correct. So, there is a network service provider. So, the one who runs a network itself there is a consumer of the network who is going to just be managing the network. Say, first provider is going to be governing the network. So, they will they will figure out who are the consortium members. How do members enter and leave the consortium?

The service consumer is going to be operating the network. So, you can think each organization having an admin, who is managing the entities in the network like peers and (Refer Time: 06:53), but the focus of composer is really in the business service provider. So, business service provider is like it could be the application developer or the architect. So, they are going to develop the blockchain application.

They are going to define what the information going to be stored on the network is going to be what are the assets. What are the transactions that are going to be performed, what are some of the applications, how are those applications going to be consumed by the user so it could be some presentation logic there and how these transactions are going to be performed on blockchain also.

There is also a service consumer. So, this might be the end users right of your applications.

(Refer Slide Time: 07:32)



So, what are some of the key concepts here? So, we will start with the end user, right is always good to start with the end users perspective of the whole thing right. So, the end user thing of it as they want to perform a specific function maybe, I am an end user who wants to transfer balances in a bank account that is maintained on the blockchain. Or I might be looking to trade certain artifacts let us say I want to the network manages the set of cars. And I want to purchase cars on the network.

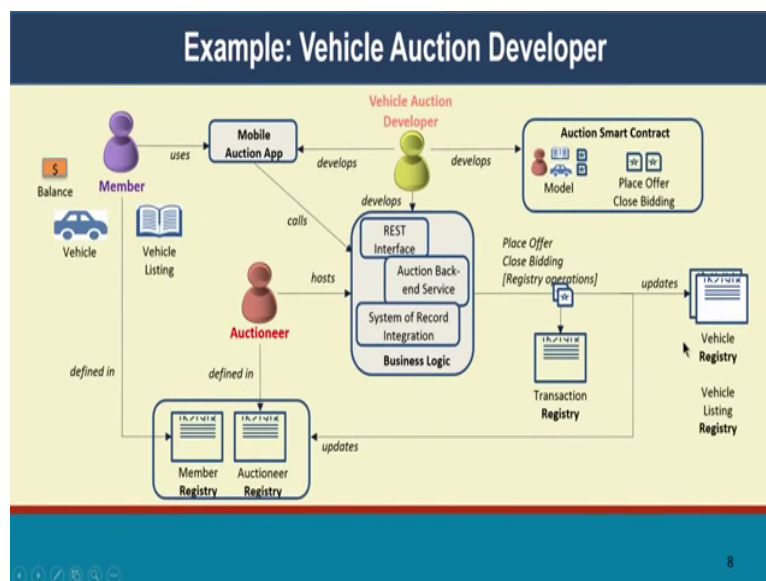
So, maybe there is a presentation logic. And the developer is going to be using that UI user interface to be interacting. Or they could also directly call API, API calls onto the onto the application you have developed. So, these end users the way they interact is they have to be registered or defined in a participant registry. So, there are a set of registered participants, and they will be part of a registry that has be defined. And who are the set of participants? The legitimate participants will be defined by a service consumer. As far they will define saying, these are set of entities belonging to my organization who should be participants of this network.

The business service provider is the developer or the architect. They are going to develop a few things right. They are going to develop the smart contract logic. So, this will define the set of assets or data models that you will define as part of this smart contract. It also defines some of the logic or the business logic encoded as functions. So, in fabric it was chaincode, in composer it is going to be node js functions. Now the business service provider apart from

developing the smart contract logic they might also develop an application on top. So, the application could have rest interface, it has some business logic, and it has some integration logic with other applications that it might interact with both internal and external to the organization.

And along with that they might also develop the UI layer for it, apart from the application. So, the end user can directly use the UI or can call the application through the rest interface. Now whenever they invoke the rest interface performing certain function invoking certain function. This invokes a transaction on blockchain. So, there is going to be a transaction registry, which tells you what are the set of valid transactions. So, this is an embodiment of the set of smart contract functions you have to find. So, there is a set of transactions you can provide you can specify. So, that is a transaction registry the transactions as I mentioned are written in node.js code. There is also an asset registry, which defines a set of assets being modelled as part of this business network.

(Refer Slide Time: 10:13)



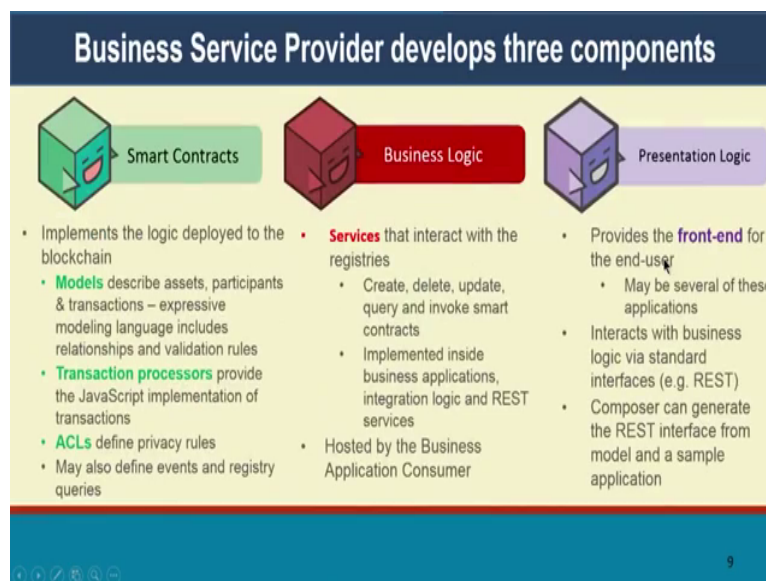
So, let us take a particular example right. Let us say this is the vehicle auction. So, who are the entities in this in this whole ecosystem? There could be a developer who is going to whole architect and develop the resolution, there are a set of members or users who are interested in owning vehicles in turns of bidding for owning this vehicles. The vehicle itself you can think of it as an asset. So, there is a so, if you of think of it and then and there is an auctioneer. The auctioneer it is an it is an optional entity in this ecosystem. They could for

instance be responsible for closing out an auction right. They might start an auction and close an auction. So, that might be what the auctioneer does.

So, what are some of the key concepts here? That you are we are interested in. What are some of the let us start from the user itself right. The user wants to see what are the set of vehicles that are opt for auction. So, there is a vehicle listing that will say these are the set of vehicles that are available for you to bid. So, each member can come see the list of vehicles, see pick a particular vehicle they are interested in and play some bid. Before they are able to list the vehicles and place a bid of course, they have to be registered in the member registry. Likewise, you can say there is a separate registry for members and a separate registry for the auctioneer. So, there could be multiple auctioneer as well.

So, those are the set of members in the network. There is a vehicle listing, and there is going to be a transaction registry. So, this transaction registry will hold what are the offers that you can make. So, when a member wants to place a bid on a particular vehicle that will be recorded as a transaction. So, that will be the transaction registry. And finally, there is a vehicle registry that has the vehicle listing; it has the listing registry as well. So, each vehicle might have attributes all of those will be stored. And when you close when you place a bid that bid will also be entered into vehicle registry.

(Refer Slide Time: 12:06)



So, that is just an example to show some of these features I will show you some (Refer Time: 12:10) of code as well to get you for you to get a feel of what composer looks like. But before

we get there some components of composer, what are the things that you will be developing as a composer user? So, the first thing is you have to develop your smart contract logic. So, smart contracts are you can think of it as comprising of the following correct. So, there is going to be a model file and this model file will describe the set of assets. So, in the previous example it was a set of vehicles, but you can think of it as any kind of assets that you want to define. And you are also going to define a set of participants who will participate in this network and a set of a transactions.

So, these transactions are the business logics that will operate on the assets. So, in the previous example, placing a bid would be a transaction whereas the vehicle itself would be an asset. Now this whole thing can is you can specify this in a very expressive modelling language that includes notions of relationship. So, I can create multiple assets and say there is a relationship between the assets. Let us say I can define let us say a member and a vehicle, and say the vehicle has one of the members as an owner. So, that is the owner becomes a relationship between members and vehicles right. So, we can define relationships like that, and you can also define validation rules.

So, on the assets themselves you can specify validation rules. And these validation rules will be automatically enforced when running the smart contract. Then there are transaction logic, the transaction processors and this is JavaScript implementation of the transactions. So, we can implement them as very simple JavaScript functions. And these will get executed in the blockchain in a decent (Refer Time: 13:55). So, we will also see how that happen. In the third thing that you can defined as access control rules. So, these are like privacy settings saying who can write which asset, who can modify which asset, who can read which assets, all of those access control rules can be specified in a separate ACL file.

So, each of these think of them as separate files that you can write and the advantage is the models can be reused across different smart contracts or and even across different networks. So, I can define one model for let us say vehicles. And I can actually use that in multiple applications. Maybe there is an insurance application; maybe there is an auction application. All of them can use the same set of models. And all these together, models, transaction processors and ACL's all taken together is what defines a business network. So, we will see that. Now apart from this developing the smart contracts, you are probably going to develop certain applications and services on top.

So, these application and services might create delete update and query some of them assets that you have defined before right. So, these are the application logic that will invoke some of these transactions to update assets. And it might also implement be implemented inside other business applications larger business applications. And it could be integrated with other systems using rest services. Then there could be a presentation layer. So, this is a front end or the user interface.

And maybe there may be several of these applications, one each organization may have it is own presentation layer. And it interacts with the business logic using rest API's. And composer has a neat way to even generate a skeleton of this presentation logic for you. And it is automatically generated code for you based on how you write your smart contracts.

(Refer Slide Time: 15:42)

Key Development Concepts

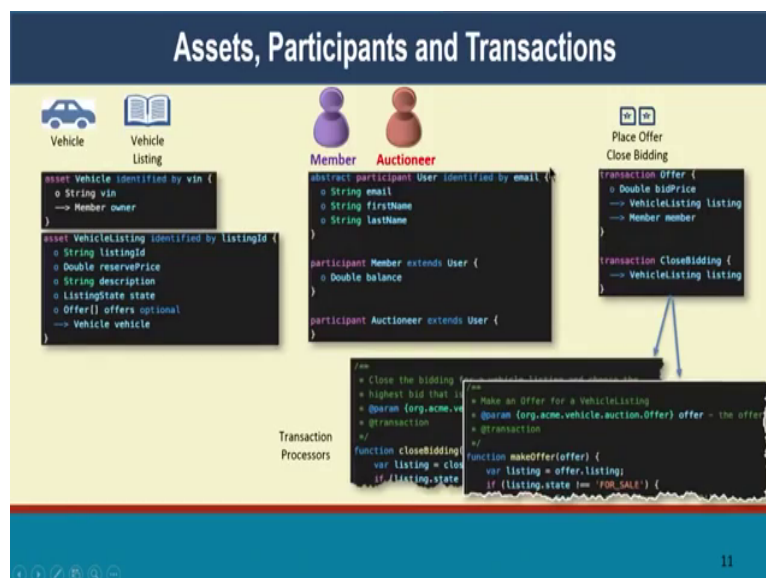
- **Model files** describe the **assets, participants, and transactions** in a business network
 - Expressive modeling language includes relationships, arrays and validation rules
 - Data serialized as JSON, and is fully validated by Composer runtime (implemented as a chaincode in Fabric)
 - Model files can be shared and reused across business networks
- **Access control lists** define rules for sharing and privacy
 - Rules automatically enforced by Composer runtime
- **Transaction processors** implement additional business requirements
 - Standard javascript code executed on the Fabric network by Composer runtime
 - Each transaction processor function executed atomically on blockchain
- A **business network definition** is the set of the above for a given business
 - Has a name and version number

So, what are some of these key concepts right we have talked about model files. The model files describe the assets participants and transactions of the business network. It is a very expressive modelling language. It includes relationship arrays and validation rules. And all of this each let us say an asset and participants they are captured as JSON so, they are all serialized data as JSON. And it can be fully validated by a composer runtime. So, think of the composer runtime as actually a chaincode a special chaincode that is actually running within fabric, which will interpret all of this that you all of this logic that you specified. Be it assets, be it participants or transactions, this runtime is going to interpret all of this information that you are providing it.

Model files can also be like as I said shared and reused across multiple business networks. There are access control lists that define the sharing and privacy. And these rules are automatically enforced by the composer runtime again. So, we will look up for each asset what are the access control rules, and it will enforce that the transaction processors require implement the business logic business requirements.

Each standard JavaScript code and it is executed again automatically on the blockchain when you are in working those transaction. All these together forms what we call a business network definition right. It is called a bni file and all these are part of that.

(Refer Slide Time: 17:03)



So, here is an example again of the vehicle auction example itself. To give you some idea of what each of these is going to be. So, our main asset is here the vehicle. So, if you look at it is define asset vehicle, it is identified by a vin. So, this is the think of it as a primary key in database parlance.

So, there it is a string. So, this is as like a standard object. And this hyphen hyphen greater than symbol represents a relationship. So, this vehicle has a relationship with a relationship called owner with a particular member. So, let us look at what member is right. So, member is a participant in the network, it is a special kind of participant that we are defining. So, first we have defined this abstract participant that has properties of email username and first name and last name.

And the member is an extension so, this extends is a standard inheritance terminology. Member extends user and it has a single element within it which is the let us say the account balance of that particular member. So, how much money do you have in your balance? Of course, you cannot bid greater than how much money you have in your balance, you cannot place a bid more than that.

Auctioneer is also a user, but they do not have any balance because they are not going to be placing any bids. So, we do not need any balance for the auctioneer. So, the vehicle could be owned by somebody who is a member of the network. Then there is a vehicle listing, again identified by a listing ID. So, you could think of multiple such listings it is a identifier listingid. And the whoever is setting up this auction can set a reserveprice. And this reserveprice is basically something let us says that if the bid is going to be less than this reserveprice, then I do not even want to sell my vehicle. Only if the bids are more than this I want to sell it so, that is the reserveprice. There is a description and there is a listingstate actually it is not shown here, but listingstate is actually of type, it is an enum variable, it is an enumerated variable. So, you can also have enumerated variables and composer.

And it is a state right the state could possibly take different values saying this vehicle has been offered an auction, it is on sale in the sense that your bids have been have been made for this. Or it could have a final state saying sold so and so forth right so that will be the state of the vehicle itself. And there could be multiple offers that are placed. So, each time an offer is placed initially it will be empty. But each time an offer is placed it will get added to the set of to the array of offers. And it has a relationship to a particular vehicle. So, each listing will be associated with a particular vehicle, again it is a relationship to the vehicle asset. So, this way you can easily think of it as business contract. So, here if you look at it I have only noted it in terms of vehicles a listing, a members and the placing of bid and so on right.

So, these are all business contracts. I do not have to there is no where here do I have to worry about whether there is a certificate involved, what peer I have to connect to what channel I have to connect to and so on right. All of that will be abstract a way by composer made easier for you. And then there could be transactions. So, this is the business logic itself. So, here we are defining what those transactions maybe, again with relationships to vehicle listing and member.

So, this is an offer that someone is going to be placing. And there are certain functions you can call. So, let us say there is a make offer. Where a particular member maybe making an offer on a vehicle, you will invoke that function to execute that logic. And what that logic will do is will first check whether you are a valid member; if you are valid member check the vehicle you are placing a listing on whether it is open for action. And then whatever is your offer, it will add that to your offers variables. So, that is kind of what the function would be.

And you can implement that in JavaScript script easily. There could be other functions; like, auctioneer for instance could close bidding, after a period of time they could just call that function to close the auction and whoever is the highest bidder could be the owner of the will be made the owner of the of the contract.

(Refer Slide Time: 21:13)

Access Control

```

rule EverybodyCanReadEverything {
  description: "Allow all participants read access to all resources"
  participant: "org.acme.sample.SampleParticipant"
  operations: RRD
  resources: "org.acme.sample.*"
  actions: ALLOW
}

rule OwnerHasFullAccessToHisAssets {
  description: "Allow all participants full access to their assets"
  participant(p): "org.acme.sample.SampleParticipant"
  operations: ALL
  resource(r): "org.acme.sample.SampleAsset"
  conditions: (r.owner.getIdentifier() == p.getIdentifier())
  actions: ALLOW
}

rule SystemACL {
  description: "System ACL to permit all access"
  participant: "org.hyperledger.composer.system.Participant"
  operations: ALL
  resources: "org.hyperledger.composer.system.*"
  actions: ALLOW
}

```

- It is possible to restrict which resources can be read and modified by which participants
 - Rules are defined in an .acl file and deployed with the rest of the model
 - Transaction processors can also look up the current user and implement rules programmatically
- ACL rules can be simple (e.g. everybody can read all resources) or more complex (e.g. only the owner of an asset can do everything to it)
- Application supplies credentials (userid/secret) of the participant when connecting to the Fabric network
 - This also applies to Playground!

12

Now let us look at some examples of how you could define an access control in this in this example right. Or more generically now it is possible to restrict. Who accesses what assets, and what rights do they have on that asset whether it is whether they are able to create that asset, whether they are able to modify that asset call a particular function on that asset, whether they are able to read it, all of those can be defined by access control tools.

It can be very simple starting from everybody can read everything. So, here is a that first step example this is for that. Everybody can read everything it is going to do it is going to have who are the participants, any kind of sample participant any participant that follows this is ok. What is the operation? Everyone can read. So, that is the read operation, what is the resource?

All resources. So, it is sample dot star and action is to allow. So, we can say deny there. So, you can say deny this particular user access to this resource so, that is also possible. So, you can have more complex access control rules as well.

So, you can say only the owner of a particular asset can do anything with it. Or can do basically they can read write modify everything alright. So, here again owner has full access. So, there we going to allow, what is the operation everything right read, write everything. Who is the participant? This participant follows this. What is the resources? It is a one particular asset. And you can specify a condition there that says owner is really this participants. So, whoever is invoking is actually the owner. So, only if this condition is satisfied will they be allowed to perform or will they be permitted to for that to perform that transaction and you can also have credentials right.

So, based on the users secret that you are using, you can ensure that only certain users are authorized to perform a certain transactions. And these access control rules are also available in the playground even if you are doing a toy. Example that you are building you can check this check these out, check these access control rules out in playground as well.

(Refer Slide Time: 23:16)

Events and Queries

- Events allow applications to take action when a transaction occurs
 - Events are **defined** in models
 - Events are **emitted** by transaction processor scripts
 - Events are **caught** by business applications
- Caught events include transaction ID and other relevant information

- Queries allow applications to perform complex registry searches
 - They can be statically defined in a separate .qry file or generated dynamically by the application
 - They are invoked in the application using `buildQuery()` or `query()`
 - Queries require the blockchain to be backed by CouchDB

```
event SampleEvent {
  -- SampleAsset asset
  -- String idValue
  -- String module
}
```

```
// Emit an event that the module asset
var event = p.factory().emit('org.acme.sample', 'SampleEvent');
event.asset = tx.asset;
event.idValue = idValue;
event.module = tx.module;
emit(event);
```

```
businessNetworkConnection.on('SampleEvent', event) => {
  console.log(event);
}
```

```
query selectCommoditiesWithHighQuantity {
  description: "Select commodities based on quantity"
  statement:
  SELECT org.acme.trading.Commodity
  WHERE (quantity > 80)
}
```

```
return query('selectCommoditiesWithHighQuantity', 0)
```

13

The other features of composer are events in queries. So, with events this is similar to the fabric events that I that I mentioned right. So, again with composer events, you can define events on the models, you can define events are emit. So, they events are defined as part of

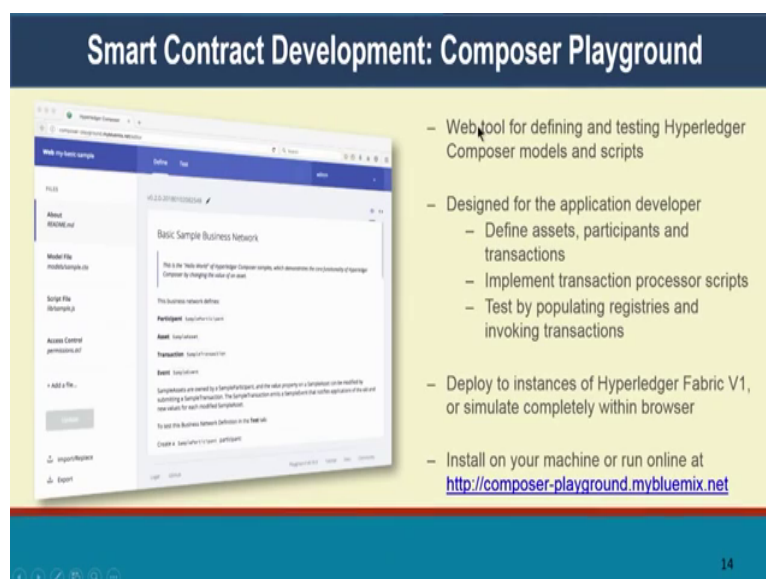
the models themselves. So, when you are defining your model file, you can say emit this model is this is a particular event defined in the model.

And based on the events defined, you can emit them as part of your JavaScript functions your transaction logic. And business applications can trap these events and perform other functions on top right. So, this is very similar to the chaincode events that I mentioned to you in fabric. The events include when the events are emitted, they also include the transaction ID and all other relevant information for you to take business business for you to do business functions on top right. So, you will have what is the transaction ID on blockchain that was part that committed this transaction that emitted this away.

Now, the other thing you can do with composer is queries. So, in hyperledger fabric we have support for both GolevelDB, as well as CouchDB as the state database. So, the CouchDB gives you ability to store elements as JSON documents and gives you rich querying capabilities as well. So, if you are using if you set up your fabric with CouchDB support, then you can do a lot of rich queries on the using composer as well.

So, you can perform complex registry queries right. So, we can say give me between this range and that range give me all the elements in this range. So, there are a set of complex queries it can perform on the assets. That you have you have that you have defined. And they can be invoked from within your application that you are developing. But the only requirement is that you have used CouchDB as the ledger for the underlying fabric.

(Refer Slide Time: 25:22)



The slide features a dark blue header with the text "Smart Contract Development: Composer Playground" in white. Below the header is a screenshot of the Composer Playground web interface. The interface shows a sidebar on the left with options like "About", "Model File", "Script File", "Access Control", and "Add a File". The main content area displays a "Basic Sample Business Network" with sections for "This Business Network Defines", "Participant", "Transaction", and "Asset". To the right of the screenshot is a list of bullet points describing the tool's features and usage. At the bottom right of the slide, the number "14" is displayed.

- Web tool for defining and testing Hyperledger Composer models and scripts
- Designed for the application developer
 - Define assets, participants and transactions
 - Implement transaction processor scripts
 - Test by populating registries and invoking transactions
- Deploy to instances of Hyperledger Fabric V1, or simulate completely within browser
- Install on your machine or run online at <http://composer-playground.mybluemix.net>

So, this is the composer playground this is a quick link for you to quickly go try it out yourself. The composer playground is just JavaScript interface it actually does not connect to it, if you have the option of connecting to a fabric in the backend. So, you can have a actual blockchain instance in the backend. But for quick quickly trying out certain things just to see how application development works. You not even need a fabric network. So, you can do it completely just simulate things in the browser. So, it gives you a way for you to test what else.

And it is a easy tool for you to develop composer models and scripts. It is define by the application developer. So, you can go online easily and you can define your assets participants and transactions. You can define your scripts for your functions and you can easily test it as well by invoking a few transactions against it. And you can also install this whole code that the playground code on your laptop. And you can do it in your at your region as well you do not even need an internet connection after that.

(Refer Slide Time: 26:26)

General Purpose Development: Visual Studio Tool

- Composer extension available for this popular tool
- Features to aid rapid Composer development
 - Edit all Composer file types with full syntax highlighting, code completion
 - Validation support for models, queries and ACLs
 - Inline error reporting
 - Snippets (press Ctrl+Space for code suggestions)
 - Generate UML diagrams from models
- Install directly from Code Marketplace

The slide includes several screenshots: a code editor showing JavaScript code for a lifecycle, an error message in the console stating "[Composer] IllegalModelException: Could not find super type Person", a Code Marketplace listing for the Hyperledger Composer VS Code Plugin, and a UML class diagram showing relationships between classes like Participant, PrivateOwner, and Poolson.

So, I mentioned earlier that we do have composer has support for multiple editors. Here I am just highlighting the visual studio tool, where composer has a an extension for visual studio, that provides easy development capabilities right. So, it is gives you full syntax highlighting for composer. So, in this case all the things like keywords like assets enum transaction they will all be highlighted for you for easy use.

And it also has abilities for code completion. So, it will automatically complete variable names. And so on it has validation support. So, while you are typing out your code actually tell you if you are making a syntax error or something like alright. So, it can will gives you validation support when you are defining your models your queries and your ACL's. So, that is the inline error reporting I mentioned. So, while you are typing out you can quickly get hints as to if you doing something wrong. You can have control plus space for code suggestions. So, you can get snippets directly. And you can also get UML diagrams and models generated automatically right.

So, this is available on the code visual studio code market place, you can download it from there and you will be of to go very quickly.

(Refer Slide Time: 27:38)

Creating the Business and End-User Applications

- JavaScript **business applications** require() the NPM "composer-client" module
 - This provides the API to access assets, participants and transactions
 - RESTful API (via Loopback) can also be generated... see later
- Command-line tool available to generate end-user command-line or Angular2 applications from model
 - Also helps with the generation of unit tests to help ensure quality code

```
const BusinessNetworkConnection = require('composer-client')
    .BusinessNetworkConnection;
this.bizNetworkConnection = new BusinessNetworkConnection();
this.titlesRegistry = this.bizNetworkConnection.getAssetRegistry(
    'net.biz.digitalPropertyNetwork.LandTitle');
let landTitle = factory.newResource(
    'net.biz.digitalPropertyNetwork', 'LandTitle', 'LID1148');
landTitle.owner = ownerRelation;
landTitle.information = 'A nice house in the country';
this.titlesRegistry.add(landTitle);
```

```
yo hyperledger-composer
```

model

VehicleListing

id	owner	location	status	price	date
1	John	London	available	100000	2018-01-01
2	Jane	London	available	100000	2018-01-01
3	John	London	available	100000	2018-01-01
4	Jane	London	available	100000	2018-01-01
5	John	London	available	100000	2018-01-01
6	Jane	London	available	100000	2018-01-01

16

Creating a end user application. So, this look as I mentioned the all the composer code is in JavaScript your transactions and all that are specified in JavaScript. For doing for writing these things and connecting with composer runtime, you need the composer client module that is available as an NPM. And a composer uses loopback to automatically generate restful API's for your assets. So, for each of your assets it automatically generates crud operations to create update delete modify these assets, it gives you a an automatically generated code for these rest API's.

And this also has a way for you to generate angular 2 code. You can generate both CLI code as well as and an angular 2 code. For you to stand up a user interface very, very quickly. So,

once you have defined your models and transactions you can use these to generate a restful API and then you can create a user interface that calls that restful API.

So, you can go list your assets, you can go create a new asset, you can modify that asset you can delete that asset. All those there is a easy user interface that is also generated in angular 2 for you so, that can be a starting point for you to develop the presentation logic or user interface for your application.

Finally, composer has debugging support as well right.

(Refer Slide Time: 28:56)

Debugging

- Playground Historian allows you to view all transactions
 - See what occurred and when
- Diagnostics framework allows for application level trace
 - Uses the *Winston* Node.js logging framework
 - Application logging using `DEBUG` env var
 - Composer Logs sent to stdout and `./logs/trace_<processid>.trc`
- Fabric chaincode tracing also possible
- More information online:
<https://hyperledger.github.io/composer/problems/diagnostics.html>

Default Historian Registry

ID	Time	Participant ID	Transaction Type	View Data
af5a01d73-6d7-504d9f5030...	17:15:00	anna	Offer	View Data
7b40833-51f-4d3-9d71-4d97...	17:14:34	nguyen	ActivateCurrentIdentity	View Data
e5d3d75-7ee0-46b-a7f938ba...	17:14:30	rust	Address	View Data

Transaction Data

```
Transaction: Events (3)
{"@class": "org.hyperledger.composer.system.HistorianRecord",
 "transactionId": "af5a01d73-6d7-504d9f5030...",
 "timestamp": "2017-07-14T17:15:00.000Z",
 "transactionType": "Offer",
 "transactionData": "TransactionData",
 "participantId": "anna",
 "transactionId": "af5a01d73-6d7-504d9f5030...",
 "timestamp": "2017-07-14T17:15:00.000Z",
 "transactionType": "Offer",
 "transactionData": "TransactionData",
 "participantId": "anna",
 "transactionId": "7b40833-51f-4d3-9d71-4d97...",
 "timestamp": "2017-07-14T17:14:34.000Z",
 "transactionType": "ActivateCurrentIdentity",
 "transactionData": "TransactionData",
 "participantId": "nguyen",
 "transactionId": "e5d3d75-7ee0-46b-a7f938ba...",
 "timestamp": "2017-07-14T17:14:30.000Z",
 "transactionType": "Address",
 "transactionData": "TransactionData",
 "participantId": "rust"}
```

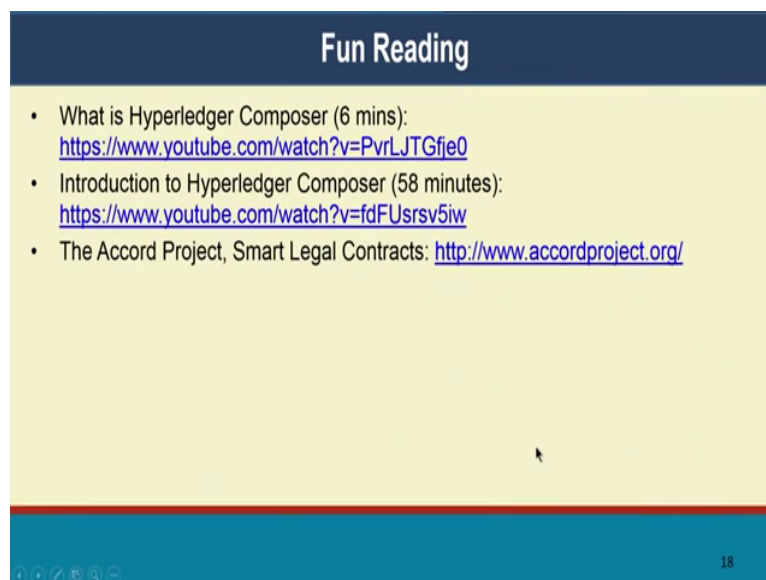
So, it has the ability something called playground historian in composer. That allows you view all the transactions you have performed. And you can see what happened when what is the particular transaction that happened will give you a full listing of everything you have. And that can be very helpful for you to debug your code. And it also gives you a diagnostics framework for application level traces. So, you can there is a it is uses the Winston node js logging framework. So, as part of your transactions you can quickly get access to the log set were generated.

So, using the debug environment variable you van setup the logs. And you can go lookup a transaction you can go lookup the logs quickly pertaining to that transaction that will help you debug quickly. And in addition apart from just your composer code or anything that you are writing in your application, it is also possible to get the chaincode level tracing. So, you

can exactly see what is happened in the fabric layer as well, and you can track what is happening on the blockchain. And all of these is very well documented documented. So, you can go lookup composer documentation and see some of these features and read more about that.

So, that concludes the first section of composer itself.

(Refer Slide Time: 30:12)



The slide is titled "Fun Reading" in a dark blue header. Below the header, on a light yellow background, is a bulleted list of three items. Each item includes a brief description and a URL. The first item is "What is Hyperledger Composer (6 mins):" with the URL <https://www.youtube.com/watch?v=PvrLJTGfje0>. The second item is "Introduction to Hyperledger Composer (58 minutes):" with the URL <https://www.youtube.com/watch?v=fdFUrsrv5iw>. The third item is "The Accord Project, Smart Legal Contracts:" with the URL <http://www.accordproject.org/>. At the bottom of the slide, there is a blue footer bar containing navigation icons on the left and the number "18" on the right.

So, there is a easy video the quick video about composer and some of it is features so, it is 6 minutes. But if you want to get started with using composer, I would definitely recommend you to both try out the playground as well as see this about one hour one-hour lecture alright. It gives you all these concepts defined, and it gives you a even a demo of all of these things. And I would also like to highlight this other very interesting project called the accord project. It was started by a startup called clause, clause dot IO.

And what it tries to do is it tries to build this notion of legal smart contact. So, what does it make for a smart contract to actually be recognized the something legal. So, it is a very interesting project, I would encourage you to check it out. It is closely related with composer I would say in terms of the principle but here they are specifically looking at legal contract. I would encourage you to take a look at that. With that next lecture we will look at how you set up a composer network itself. So, see you soon.

Thank you.