Design of workflow system for business process intelligence support

Milan Pospíšil
Fakulta informačních technologií,
Vysoké učení technické v Brně,
Božetěchova 2, Brno

Abstract: This paper deals with part of design of workflow system for support of business process intelligence tools – mainly monitoring, simulation and optimization. Part one is brief introduction of our goal, but deeper informations can be found in my previous work of Business Process Simulation using Business Process Intelligence. Second chapter deals with desing of workflow system, whitch deals only with control-flow, services and data. Full design of system including resources, simulation, monitoring and optimization is beyond this paper.

Content

1 Introduction

This work is about two main parts of project. Chapter 1 is about our goals, about Business Process Intelligence, simulation and optimization. Chapter 2 is about design of our workflow system.

2 Goal of work

We have some goals in our work, that are described more in my previous work in encosed document Business Process Inteligence. But here is conclusion:

Our work deals with Business Process Intelligence, whitch is composited with these areas:

- Analysis — Offiline analysis of historical data.
- Monitoring — Online analysis of data, realtime reaction.
- Optimization — Based on observed data, we can design optimization.
- Simulation — Simulation can be used to optimization or prediction.

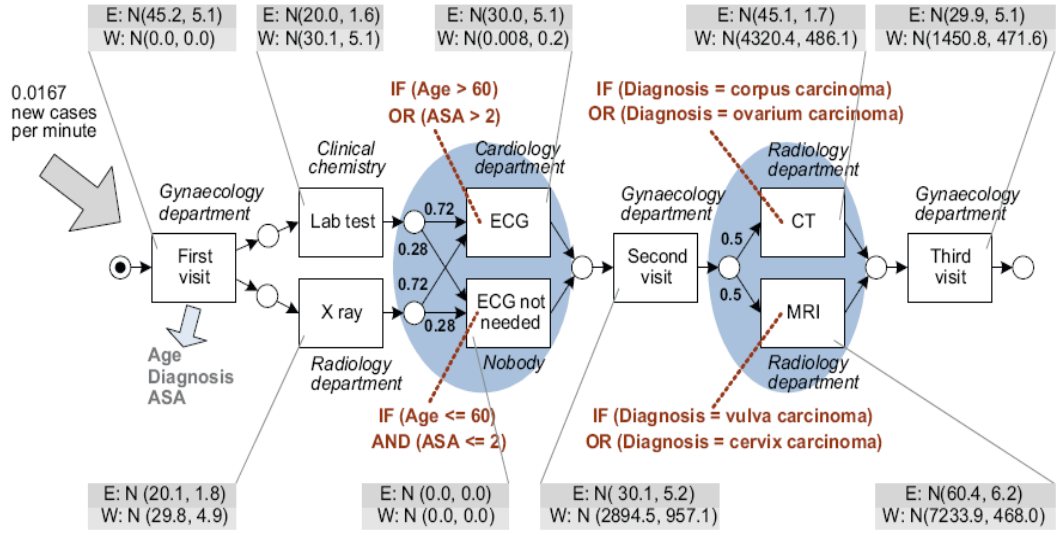
All of these areas are closely bounded. Monitoring uses data from analysis and can use some regression, decision trees, or fast forward simulation for operational decision support.

Simulation model can be built manually, whitch is time-consuming and these models are not so precise, or from analysis of historical behavior. Then, simulation can be used in Monitoring, for prediction, or for optimization.

Optimization can be done directly from some data performance analysis, and some changes can be done by searching in space of possible solutions and this solutions can be measured by the simulation. Then fitness of solution is multicriterail function, and its evaluation is based on managers demands, for example we are interested in price, quality of services, time, or risks. Risk can be defined by some very distinct bad results in repeated simulations, whitch are normally good solution.

So our work will be heavily based on collecting behavior of workflow system and then offer some advanced functionality described previously. Some work has been done by W.M.P Van Der Aalst and his team (citations are in the enclosed document Business Process Intelligence). But this team did not went to far, they only built classical simulation model using datamining. We want to go further and use as much data as it can be possible.

Here is example of classical simulation model:



We can see, that all is described in classical statistical distribution and routing pahts are implemented by percentages or by decision points, whitch are used when we can have data for decision. This is not enough for optimization and poorly for simulation for operational decision support. This type of simulation is dealing with current state and tries to predict future behavior, that longs from several hours to several days – based on repeatability of process. We can catch some future risks using that. Next text describes bottlenecks of this approach.

2.1 Mining deeper dependencies

Now, we begin our own short discussion about pitfalls of current researched simulation models for operational decision making. We belive, that described models are still not sufficient. Realize, that we are dealing with simulation of near future with current state of business process, with ongoing cases and current resources available. This is very important fact, because current models parameters are estimated from long running time. The current situation is now only small subset of modelled behavior and can differ significantly from the average behavior. But for operational decision making, it is very important to model just that extraordinary behavior, because manager will not be interested of future of well known ordinary process state. Note that extraordinary means still inside the boundaries of mined data.

But that more precise approaches stands and falls with quality of data. So this is now more dream, that reality, because we do not count, that companies have all data needed for that type of extraction. But if the simulation using process mining will be found perspective, they can record more quality data.

We will now discuss some pitfalls:

Execution time of Task

Execution time can differ for different case parameters and particular resources can also have influence, because some workers are more productive than others. For example, we can have repair order with parameters, and task, that repairs the product. We observed, that execution time is significantly influenced with some paramers, like product type, age of product and resource. Some experienced workers could repair it much more faster, than others. If we have sufficient data in cases, we can mine that dependencies, and if they are significantly different from average parameter, we can add exceptions.

Execution of task can be influenced by many factors. Not only the case parameters and resources, but also foor example on time and weather., because some external services are more available in certain time of day, shipment of some needed goods can be faster when there is better weather. That is maybe too much level of detail, but if the process heavyly depends of it, it can be wise to add that informations to cases so data mining can find dependencies there.

Probabilites of routing are not enough for short-term simulation

In operational decisions, we have cases in our system, so we know their data, whitch are used for routing (only if they are). So routing path is heavyly influenced, probabilities are now useless. Probabilities can be now only used to model some new incoming cases. But managers could also want to know, what will happen if some special cases arrives (for example, because they predict, that thay could come), now they could manually insert that cases and watch the influence.

Case arrival rate could be poor abstraction

As we said about execution time of tasks, the same thing could be say about case arrival rate of cases. Cases can be very influenced by particular parameters, like time and date. Some cases type are more expected only in season, in certain daytime etc. We belive that dependencies are important for short term simulations and they need to be added to model.

All mined informations are dynamic

Be carefull of changes. Not many process mining methods deals with changes. For example Decision Point Analysis can mine decision rules. But when some change of rule occures, it is useless. So mining approaches must be ready for that, and when some significant change of data is here, it must be found. Decision point analysis then can mine even evolution of rules, cases dependencies must take account to time, because that dependencies are also dynamic and more late informations need to gain more wegth. Mined model must be automaticly checked all the time and learned from new data.

3 Workflow system design

Today, many workflow systems are at the market, but optimization and simulation may require too much change of some chosen system. For example, simulation model is created from historical data, so many new informations may has to be logged. Optimization needs process changing on the fly, whitch could be problem. We decide to develop our own system, because changing of some existing system could be more difficult, than developing of whole new system.

3.1 Development platform

We choosed .NET framework for developement, because its well-known platform usable for Rapid Application Developement and its still quite fast. It has problem with running on Unix, but this is not so important for us, because Workflow system will be built using principes of web services and web client. Maybe Microsoft Silverlight [1]will be used as richer user interface, if we will be sure, that it will run on Unix Browsers without problems. Current implementation of .NET Framework runs on Unix using Mono virtual machine, now, it has some problems with newer .NET versions. But we assume, that these problems will be smaller after several years of working on project. Another important fact is that we have good experiences with .NET.

3.1.1 Scripts

Workflow system also needs interpret language to some system modifications without restarting system. Good example for this is decision points based on data. But scripting is much more powerfull. It enables us to work with workflow objects, data, control-flow and resources. On the other hand, scripts are error prone and for difficult operations are not reccomended. But there are several ways how to overcome that.

We can use web services and implement some work using service program. This is good way, we can change service at runtime, debug it and it could run on different platform. But this does not enable us same functionality as scripting. We can only call it, pass some data to it, and then wait for result. This approach can profit from connection with scripting, because sometimes, we have to do some untrivial trasformation of data, that needs to be passed to service and scripts could be very good for that. Script can also maintain executing service calls and waiting for result.

Second way is to use some plug-in in form of user dll, that is added to the application. Then, the code can be called from script. Good for writing and debugging, it also enable us to reach same functionality as script, but it needs restart of system.

---

[1] Microsoft Silverlight is similar to Flash, but it is using .NET framework and it is mainly for data applications, not for graphics and animations. Advantage is good communication with .NET server using Windows Communication Foundation.

Third possible way is to use Dynamic Language Runtime (DLR) of .NET and dynamically add some classic .NET code (not script) to the workflow. DLR also enable us to drop this compiled code at runtime, and add new version. But we don't know much about that approach, so we cannot say now, that it could work the same way, we think it works.

All of three aproaches do not také into account one thing – versions. If some case still runs and the code changes, it is maybe desirable to use old version. Scripts are automatically prepared for this, because all running cases have old copy of every component functionality it uses. And new version is only loaded if component is explicitly asked for this. Note that copies of scripts could be space-consuming, but not i .NET. .NET automatically places all strings into the pool, so there cannot be equal strings, all equal strings references to the same string in memory. Only problem is, when you need to change string, new string has to be created, but script changing will be sporadic action.

Python and its .NET implementation **IronPython** has been chosen as scripting language, but other languages could be used, if there is good support for them for .NET. It works easy way, script engine creates new scope, that needs only two informations:

- string as a script
- list of pair key, object reference, whitch represents objects, that are passed to the script

So if we need to solve security, we may allow to enter only some data of workflow engine. This, in future maybe need some architecture changes for security...for example, we can built some user classes that reflect some functionality as base classes, but without threat references to some parent instances, that can allow browsing to all instances. Another solution can be in security support in .NET.

3.2 Abilities of system

Current system design (and partly implementation) contains these things:

- Control-flow (Sequential, AND, OR). Complicated routing can be written using scripts.
- Data (Case data, Task data and Workflow (global) data, possibility for data passing by value, or by reference – only in the same case or script, not in external case or service).
- Scripting.
- Events and exception handling.
- Calling web services via data passing.
- Calling C# code from script using plug-ings.
- Persistence of whole application (saving state to database or file possible every run step and restoring it when application starts).
- Logging changes for analyze and history replay for user interface.
- Recursive case calling using data passing.
- Parallel running of cases (every case in one thread).
- Definition change implicitly doesn't affect running case instances.
- Support for changing case definitions at runtime only for one case (Ad-hoc process changes). Those changes can be then saved as new process definition.
- Possibility of distributed computing with several Workflow Servers running on different computers communicating using Windows Communication Foundation (similar to web services, faster than web services, not using xml, using binary persistence). Note that distribution will be only over cases, shared workflow and resource data will be stored only in one place.

3.3 Main components

There are three main types of components:

Workflow
Definitions
Instances

Workflow contains shared things like workflow global data, resources and information about definitions and running instances. Workflow component can also communicate with other workflow systems, when running in distributed enviroment. Workflow component always runs in one main thread. It is singleton, so its available in the whole process.

Definitions are handled by workflow component. Definitions can be case definitions, control-flow definitions, some resource definitions, scripts…etc. They are all defined in shared space and all definitions have **Definition time.** This is very important for versioning, because when case loads some definition at runtime, it choses the definition that was valid when case started. Of course, every definition reference can contain settings, that says, that we want always use newest definition. Old definitions can be deleted, when there is newer verison and there are no running cases that runs with time, when the version was valid.

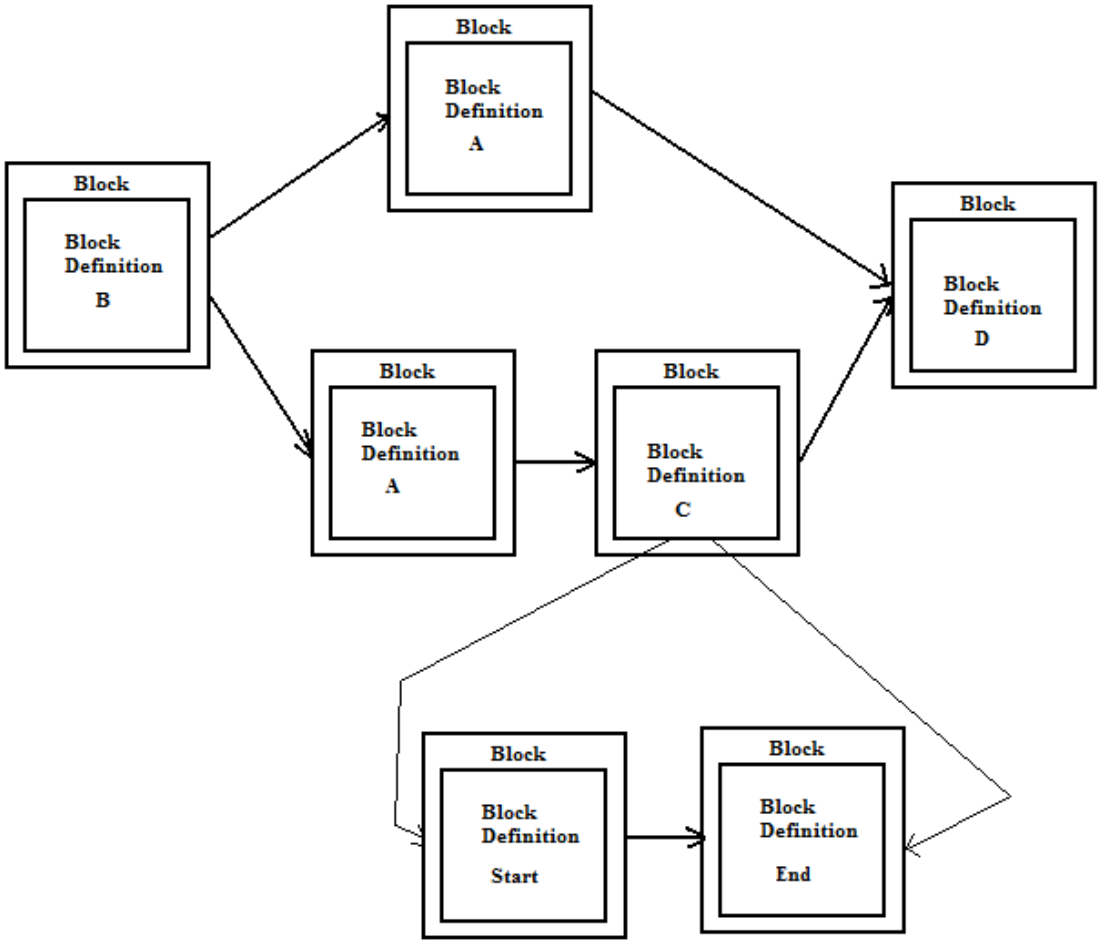
Instances represents running instances and their instance data, now only cases. Every case runs in one thread. Cases also have their own copy of definition. Yes, this is memory-consuming, but it isolates whole case when running from outside changes, making them able to change itself and save that definition as new case definition, when it is suitable. Another advantage is that we don't need to lock definitions, because they have copy and that copy is only accessed from one thread. Note that there is possibility tu run scripts in separate thread (asynchronous calling), but those are scripts that are executed using data passing by value and they cannot change anything in process.

3.4 Control-flow components

There are several important components of control-flow

BlockDefinition	Definition of some task, scripts, subnet, task data, needed resources. Block definition is one of the most important definition component of whole system. Block definitions can define subnet – that enables us to use hierarchical nets with shared data without data-passing. Another option is to start new case, pass data to it and wait for result.
Block	Block represents Task. It uses block definition (every block has exactly one block definition) and defines connections to other blocks, and also routing (sequence, and, or., custom…). Fact that the block and block definition are different components makes use of duplicit tasks, change behavior only by replacing functionality by one block at runtime without need of reconnection and. Etc.
BlockInstance	Block instance hase close connection to block component. It contains data of one concrete running block. When case is starting, BlockInstance is created for every block makes 1:1 relation. This divide brings maybe little more compications in implementation, but it enable us to differ between definition and instance data (current state of block). It enable us to change block and then easily save it as new definition without BlockInstance runtime data.
Case definition	Represents definition of whole case. Case contain exactly one Block, that defines all behavior of case, because this block can have subnet.
Case	Instance of case using some case definition. Case have copy of its own definition and it contains all block instances referencing to all blocks.

This picture shows relations between components:



You can see, that Block Definition A is used twice as duplicit task. Block definition C uses subnet. So, if duplicit A will be using subnet, these subnets will be here twice. This is good for clarity and simplicity, because all subnets are running in the same case, thread and with the same case data – althrough we can create subnet data, so block in subnet will use these, not data of main block of case. For more independent networks is recommended to use it as whole case using data passing, but both approaches are similar. We will discusse these two another time.

Also BlockInstance object will be created for every Block object. These BlockInstances are not owned by blocks, because they are not part of Definition persistence schema, this allows to save definition without BlockInstances. So Case owns one BlockInstance and one Block, main BlockInstance owns subinstances that reflect the defined subnet. When case is load from persistence, all BlockInstances are connected to coresponding Blocks.

3.4.1 Persistence of Instances and definitions

Persistence is very important, because it enable us to save and load workflow state and to send components through the network. Persistence could be very time consuming to implement, but not with **Reflection**. .NET is a platform that exposes metadata, that can be read runtime and using that, we are able to save any object, that we annotated in our code. We must do only one thing – annotate references, that we want to persist. That means, we must say what object references current object owns and its responsible for its creation and destruction. Then, persistance can be done automatically. It is also good tool, when we want to visit all objects and do something with them. We don't have to discover circle references if we visit only references that are annotated as owner.

For persistence purposes, hierarchy of components owning (Parent-Child relation) must be defined:

- All definitions (except copies stored in cases) are owned directly by main workflow component (singleton).

- **CaseDefinition** owns **Block** component, whitch represents main block of case, whitch defines BlockDefinition with subnet.

- **Block** component owns **BlockDefinition.** This reference can be by name, that means behavior of block is defined outside and it has not own it.
  Or it can own it, that means we have inline definition of block behavior, that cannot be referenced outside. But when Case instance gets copy of definition, it also copyies definition by name and that becames inline definition too – its for better isolation of running cases.

- **BlockDefinition** owns definition of scripts and subnets. Subnets are made from **Block** components, so BlockDefinition can own **Block**.

- **Case** owns **CaseDefinition**, whitch is copy of global **CaseDefinition.** And it also contain **BlockInstances**, whitch are stored in the same hierarchy as blocks. Then BlockInstances are connected by references to particular Block components, when workflow is loaded from persistence.

So instances of cases makes copies of all definitions and also owns BlockInstance with all instance data.

3.5 Data

Data are modelled by one class, that is looking like that (this definition is not complete and its only for illustration):

```
public class DataStore
{
	// Substructures
	List<DataStore> Data;

	// Key of this record
	string Key;

	// value of this record
	object Value;
};
```

Data is collection of another DataStore objects. So it can be used as structure (data are items of this structure), or as collection where key is used as a OID of record.

This class have indexer, that is using like that:

```
Data[“Customer”][“Name”].Value = “Milan Pospíšil“;
```

The indexer works that way:

- First it looks into the Data store collection for item with key „Customer“. If it is found, it returns DataStore, that contains customer data. If it is not found, it is created, so there is no NULL reference exception.
- Then searches for „Name“ item, same way as previously. Then the user calls Value property, whitch says, we want to set value of that variable.

Value can be string, integer, float, datetime…but only known basic .NET types, standard serialization does not work with user classes, if there is no explicitly defined known types in code, whitch cannot be done in plug-in. This can only be solved with own serialization, but maybe, there is no need to use another types that basic. All types can be simulated by data structures definined in DataStore and this architecture makes it easy to transform it to XML.

3.5.1 Generating of change log

Value of DataStore (see previous definition) is also not ordinary variable, but property. Property looks like variable, it is used in code like normal variable (except references), but internally, it is method, whitch defines get and set methods. Set method generate record to **change log**, that defines data changes.

Example of data change log record:

Data Change: \Main Block -> \Warranty OldVal: No NewVal: Yes

This text representation of log item shows data change in Main Block with data path Warranty, whitch defines hierarchy in DataStore, from No to Yes.

Also, other things are logged. For example Initital Process definition, control-flow, or process changes. Log is very important for user interface and using it, we can replay process execution and show important information to managers. Another important functionality is for Analysis.

3.5.2 BlockInstance and data

Every BlockInstance can contain data. If some script want to reference data, it must reference BlockInstance first.

Types of data:

Case Data

Case data are stored in main block of case. If any script belong to the some blockinstance wants to access main data, it searches in net hierarchy (starting with Block that contain the script) to the top BlockInstance, whitch represent Case main instance.

Subnet Data

Subnet data are stored in BlockInstance, whitch owns subnet. Case data are many time also subnet data. Using subnet data, we can simulate recursive case calling in one case. Instead data passing to new case thread, we can only set SubnetData (the same way as we have to set data passing) and the whole subnet will reference those data.

But this approach is usefull only for easy problems, for more complicated recursive calls is maybe better to use recursive case calling.

Task Data

Task data belongs to the one BlockInstance. Task data are also subnet data to its subnet, but they can be used also as a storage for the scripts. Experience will show, if there is need to create TaskData different from data described below.

Data Passing inside case

Our workflow system can also support not only shared data but also data passing between blocks.

3.6 Control-flow handling

There are four basic routing types

- Sequential
- AND routing
- OR routing
- Custom routing

Fist three routings are hardcoded in the system for simplicity, the last one is completely in hand of scripts. For example, we can write script that si able to implement synchronizer control-flow pattern, or something like that.

Then, control-flow is handled in one BlockInstance that way. Flow can be in one of these states:

Input
Flow is in the input to the block. For AND routing, all inputs must be ready to continue and those are then merged to one.

Inside
This states reflects, that control flow is inside block. If block owns some subnet, control flow is then passed to the subnet and another flow is passed to **InsideSubnet** state as a information, that this block owns subnet with granted control-flow.

InsideSubnet
This is only information state, that flow is in the subnet. This state waits for returning from the subnet.

Leaving
This state says, that we are off from the subnet, and we want to leave the block. If there is OR or CUSTOM routing, we need script to continue further. In AND-routing, multimple control-flow objects are created and put to the output. Note, that script can reroute control-flow even to block, that is not connected, but it must be inside same subnet.

Output
Last state of control-flow. We can call scripts here, that can hold output (some synchronization after new path is resolved), write some addional logging information, or redirect it for some reasons…

3.6.1 Customization and control-flow events

All states, except InsideSubnet creates event, that is passed to the block. Any service inside block can respond to that. There can be also order of exection on events defined. Typically, scripts of routing are bounded to Leaving event. Some maintenance scripts can be bounded to **Inside** event.

Scripts (or another services) can control routing several ways:

- It can change the target of flow by redirect it to another block
- It can hold flow item in the block, so flow item will be stopped. This is usefull for synchronization, or for some waiting for the service, or some another event.

Note that current workflow desing is rather basic framework for future developement. This is because we now dont know what will be exactly demanded in future and so this is quite opened implementation.

Services and data passing

Service can mean many things, but they have so many properties common, so every service is handled by similar way. Services are handled by some blackbox with inputs or outputs, that may or not have access to inner data of task and case itself.

3.6.2 Services

Service can be:

Calling of another Case

Another cases are treated as standard service, with input and output data.

Calling of script

Script can be for example IronPython script. Scripts are usefull mainly for routing rules, some custom implementations of behavior (in the connection to plug-in dlls) and for data transformations – we will desrcibe it further.

Calling of web service

Web service is similar to case, its some blackbox that is doing someting, gets input and returns some output.

Standard user interface

Standard user interface is usefull for some administrative filling of data. There is need to develop some tools for automatic generating of user interface using metadata annotation.

Extern service execution

This service allows for execution of another service defined in another block. This allows for code-sharing. Its easier to define all calling and transformations in one place and then reference it from other place.

Send Mail
This allow for mail sending.

Timer
Timer is able to execute some another service (even defined in another block), but after some time.

Note that all services can be quite easily called from script, but script can make addional functionality.

We can execute it in several ways:

Synchronous call

Case thread will execute service, and wait for result. No operations are handled between that. This is usefull for small scripts, that are handling with some system wariables, for example some custom building blocks, like synchronizer etc. This is important, because of concurrent access.

Asynchronous call

Case thread will execute service, but no waiting is here. Another operations can be handeled, but task execution is waiting for call result. There is risk of some concurent access to same data, but that is problem of architecture. Note that:

- No manipulation of control data is allowed here.
- Only user data can be passed.
- Workflow data and resources are handled by Workflow core and that is using locking, so this is protected from the concurrence access.
- If there is some case data, that have to be locked, we can copy it into the workflow data repository, and now we can access it from all the workflow.

Paralell multiple call

Sometimes we need to call multimple services asynchronously, but to hold execution only in one block. So there is option to synchronously execute multiple services and waits for end.

Active x Passive call

This can be combined with synchronous or asynchronous call. Active calls are called now, but passive waits for service – good example is standard user interface.

We have to put these approaches into discussion.

3.6.3 Data transformations and passing

Transformations are need to pass data to the service, and read from it. Some basic transformations can be handled by standard methods, whitch we must consult in future, more complicated transformations can be handled by scripts. For example Age and Birth date.

Standard transformation is very simple. The definition is like that:

```
class DataTransformation
  {

      public DataStoreReference       DataStoreReference { get; set; }

      public DataSetting              ParentDataSetting { get; set; }

      public List<DataSetting>        ChildDataSetting { get; set; }
}
```

DataStore reference is reference do any data store in Case written as global BlockInstanceName. This structure looks as adresser file system in operating system.

Example: MainTask\Repair Item\Repair Advanced

Now we have block reference and the same way we can reference the data.

Example: Customer\Address

This will take Customer record and its child record Address data with its all subaddress. We can then set some attributes to the data. Attributes are hierarchical, so we can set some atributes to the root data, then child settings.

These settings are:

```
public class DataSetting
   {
      public string AccessType { get; set; }

      public string VariableName { get; set; }

      public string Availability { get; set; }
   }
```

AccessType can be:

- ReadOnly
- WriteOnly
- ReadWrite

If field is marked as ReadOnly, the user can still write to these fields, but when its returned to the server, these fields will be not copied.

If field is marked as WriteOnly, no copy will be send to client, only null Value, but user can save here some value and server will copy it.

VariableName can be used in situations, where we need to rename variable. If Empty, standard name in DataStore is used.

Availability can have two values, Available and Missing. This field is required because in Parent setting, it is said, if we want all child nodes included, or implicitly not included. Then child nodes will be makred as Missing (if parent is marked as Available).

For more harder transformation, we can use scripts.

3.6.4 Output Data

Sometimes, we have different output than input, for example when we are calling web services. So we must define output transformation from output data to our data store in very similar way, we did it previously. But in this case, we have some changes. Now AccessType is now worthless, because output data are selected, or not selected, AccessType is reduntant information.

3.6.5 System Data

Also system data can be passed, but only to scripts, because scripts are only tool, that allows for direct manipulation with inner variables and methods. Three types of system data can be passed:

FlowTarget

For redirection of flow, when we are leaving OR-split task.

NodeInstance

We can hold flow using it and wait for some event, or timer.

BlockInstance

This allows access practically to whole case net.

All system data can be passed by value (like normal data), but also by reference. Yes, it is easy to pass them all by reference, but values passed by reference cannot be tested during development. Note that BlockInstance can be passed only by reference, because its not an ordinary small structure like previous two.

3.7 Distributed Workflow servers

Every case is running in its thread. Workflow Server also runs in thread. But all is running in one process, so all cases must run on one computer. Maybe, this could be solved this way:

- We enable to use unlimited count of Workflow Servers. One Workflow server can handle multiple cases.
- One Workflow Server become main workflow server and it will hold all shared data (resources, definition, workflow global data).
- Workflow Servers will be connected using network.
- If all calling to Load and Safe Workflow objects will be handled using interface, we will be able to implent that in future, but we must take accout it in desing.

3.8 Events and exception handling

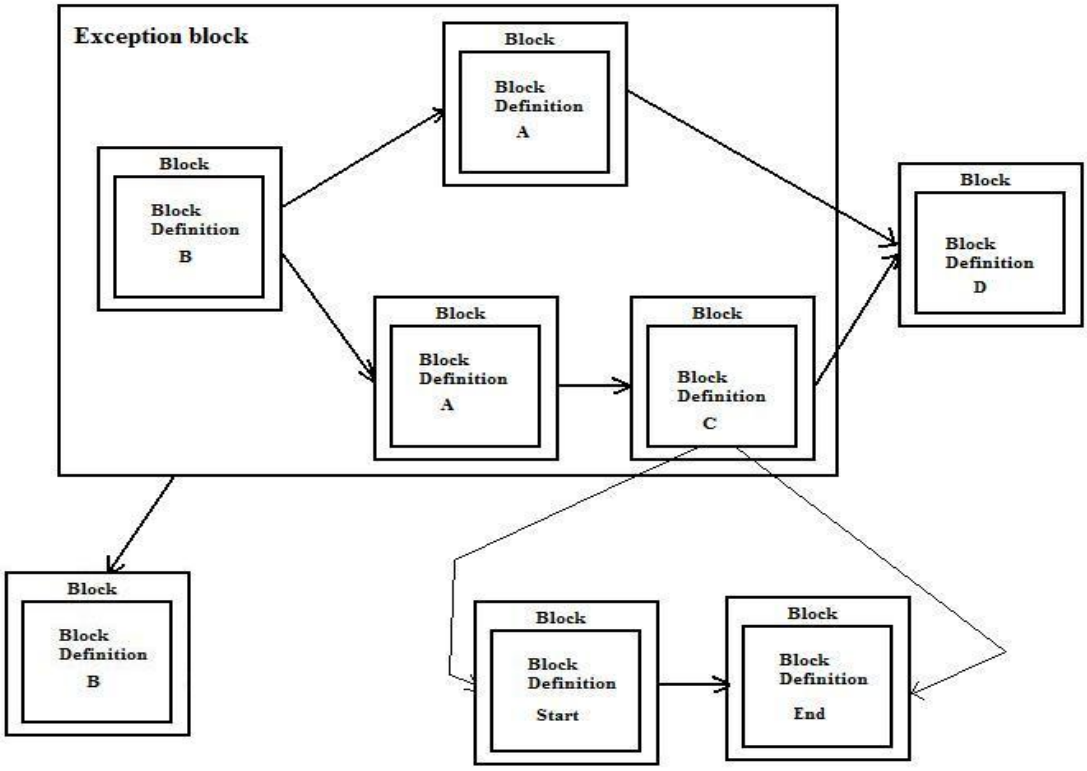
Events are now in the state of discussion. Events are triggers of services. When event arrive to the block, the block looks for the all services and executes those, that react on that.

Three types of events exists, althroug there are handled by similar way:

- **Flow events**
  Flow events were described in Control-flow handling chapter. Flow events are standards and are send only inside to block.

- **User events**
  These events can be passed in the whole network, into the subnet, or out from network to the parent.

- **Exceptions**
  Exceptions are treated as user events, but they are for signaling, that some bad things happened and we want to cancel execution and continue on different place. These blocks can be covered by special block, that is only usable for catching exceptions. When exception occurs, script at this block is invoked, and stops execution, creates new flow objects and pass control further.



3.8.1 Hierarchies of events

Events can be placed into hierarchies, so when were waiting for some event, we will react also to the events, that are more specific that our event.

3.9 Resources

We are not decided yet, how resources will be modelled in our system. Modelling of resources is not well documented of papers and our research will be closely connected to resources.

3.10 Failure recovery

We are not aware of work, that describes completely failure recovery of WF system, like Database systems have. This is because WF deals with distributed architecture and applications, that do not have transactions bound to WF system.

Our system have all state in memory, maybe only log will be stored directly to database. When log comes to some check-point, whole case state is saved to database in transaction. And when system fails, application could be loaded from database and it can run again. But there are some problems:

- Services that updated case data must be redo again.
- Services that updated global workflow data or extern data may or mustnt be redo, but system cant know that, because there is no guarantee, that external service done its transaction. Human expert have to decide this.

Now, we have only basic support for failure recovery and more work need to be done.

4 Conclusion

System design needs only shared data, resource modelling and some user interface to be fully functional. Then, tools for monitoring, simulation and optimization need to be designed and implemented. Current design is prepared for that, logs are generated for many events that happened during execution, execution times and resource parameters will be measured and stored in resource profiles. Then a model can be built using datamining technuqies. This model can be used for some detection of bad situations using decision trees, for simulation models and for optimization.

Only one problem will be failure recovery, whitch is hard in workflow systems and even commercial WF-systems are not god enough. Failure recovery will be only basic, advanced recovery is beyond our work.