Proceedings of the 11th International Conference on Applied Informatics
Eger, Hungary, January 29–31, 2020, published at http://ceur-ws.org

A Study on Formal Consistency Evaluation of Backend and Frontend Business Logic in a Modern Client-Server Application

Zsigmond Máriás[a], **Bálint Molnár**[b]

[a]Eötvös Loránd University, Faculty of Informatics, Department of Information Systems
zmarias@inf.elte.hu

[b]Eötvös Loránd University, Faculty of Informatics, Department of Information Systems
molnarba@inf.elte.hu

Abstract

The leading platforms in modern software development for applications with the user interface are web and mobile platforms. In the earlier days, it was common that the purpose of the client side of these applications only display data and process interactions, while the backend ran all business logic in the background. Recently a new trend can be identified that the client apps include complex business logic as well. Modern web frameworks and mobile technologies rely on the client device's computing capacities for the maximal performance, therefore provide built-in services to cache data and run steps of the business logic on the client device. This paradigm is illustrated by the success of PWA technology and by offline applications. The user interface of modern applications is designed by UX designers, who focus rather on the ease of use and user experience then safety and soundness, which often leads to disrupting the well-defined business process by introducing new sub-steps and alternative ways. These tendencies altogether result that the client-side of modern applications also implement a lot of business logic that already implemented by the backend in a more strict, straightforward approach. In a more abstract approach, the client and the server work with the same type of documents by performing two different business processes that exist at the same time. The two layers of the application are often developed by different teams, where frontend and backend specialists work simultaneously. Altogether this results in new risks that should be mitigated by continuously evaluating the consistency of the two business processes during the whole development life-cycle. This evaluation should identify if the two processes

are consistent, special tools have to be introduced that ensure that after performing a particular iteration in the development roadmap, the consistency is still maintained. The goal of this paper is to introduce a formal model for this evaluation and introduce tools that can be derived from this model and can be used in software development and testing.

Keywords: Software technology, Formal models, Web applications

MSC: 68N30

1. Client-server web applications

Client-server architecture is the most common and most important approach in modern software applications. The majority of user applications follow this model of architecture. The reason lies in the currently wide-spread web and mobile platforms opposite to those of desktop applications of the early 2000 years. Most of the business, communication, and social media applications are web applications where the back-end system naturally organizes data exchange of the apps on the central database.

1.1. Evolution of web- and mobile applications

Early so-called 1.0 web applications worked as thin clients as no code was running on the client-side, and the client displayed the HTML mark-up and forming command provided by the server. As a consequence, applications were hardly responsive because all user interaction resulted in a reload of the whole website. [1]

The situation has generated the demand for programs running on the client-side, therefore Javascript support was added to web browsers. Javascript support and and the AJAX [2] technology was a huge step forward. The code running on the client-side could initiate HTTP requests and communicate to the server without reloading the user side after every user interaction. Website reloading was limited to those cases when the user accessed completely different functions.

As a further enhancement of AJAX technology, modern client-side frameworks were developed which were capable to build Single Page Applications (SPA). In SPAs after initially loading the client-side application, the browser uses standard APIs (RESTful, GraphQL) to communicate with the server, and the client side logic maintains the DOM elements. The first such frameworks were Backbone.js, AngularJ, which were built around MVVM architecture[3,4]. Nowadays, the most popular frameworks as React, Vue.js, Angular2+ follow the Component Based Architecture [5,6].

For mobile applications at an early stage, software tools, and frameworks (iOS / Objective-c, Android / Java) were available, which enable them to run complex business logic. Currently, Swift programming language in the iOS environment and Kotlin in Android are the mainstream tools for application development. Both

follow the MVVM architecture, and both have API support to connect to the back-end.

1.2. Distribution of Business Logic in Web and Mobile Applications

The mainstream approach in original client-server applications is the fact that data and business logic is implemented on the server side. As to run business logic on the client-side is a necessity for several reasons, modern web and mobile applications challenged this model.

One of the typical reasons is the goal to save computing capacities on the back-end load. If all business logic is implemented on the server side, the load on the back-end increases together with the increasing number of clients. It is logical to distribute some of the tasks to the client-side as the available computing capacity and memory is constantly increasing. In the most advanced solutions, the server benchmarks the clients and based on the results decides whether to perform a computation (e.g., sort a data set) or to assign the task to the client. [7]

Another typical reason is that in modern applications development the user experience is also in the focus on top of the basic task of implementing a certain business workflow. [8] User experience creates several requirements like speed, easy access, fault tolerance and intuitive usage. Very often, the computation is made on the client-side to eliminate waiting time for the back-end communication. Typical example sorting an incoming table of data on the client-side while it is surely available on the back-end. [7]

Another use-case is when a complex process on the server-side is deconstructed into elementary steps, which are easy to understand hence easier to use for the user. The typical scenario is to divide a complex form into a series of simpler forms. The goal of this approach to decrease the probability of error by the user and to increase conversion. Churn in a process is lower in the data collection process when in each step only one question is shown, while the total amount of the fields remains the same. [8]

2. A formal approach to business logic distribution

In the previous chapters, we discussed the most critical cases informally when a web or mobile client – from now on just client – implement business logic, and identified the typical transformations. In the next chapters, we will provide a formal approach. The goal is to provide a precise tool to transform a regular business logic graph that does not take into account the client-server distribution into another extended business logic graph that specifies this aspect.

Ultimately, the goal is to construct a workflow model, from which the specification can be deducted for the backend and frontend software development, and also which helps to recognize if a backend and frontend specification is consistent.

2.1. Business logic definition using document oriented workflow graphs

Business logic modeling with workflow models has been an active field of research in the last few decades; therefore, several models are available. In this study, we propose a document-oriented workflow graph approach.

2.1.1. Workflow graphs

Workflows are usually represented by directed graphs, where nodes represent activities to perform and directed edges define the control flow, usually an edge from activity A to activity B indicating that A has to be completed before B and that some of the output of A is the input of B. There are several models available to define workflows, for example UML state machines [9]. As in this study we handle nodes as simple programs that run when the control flow reaches them rather then states in which the system waits until the next activity, therefore will use and extend a workflow model introduced in earlier research and will focus on the relevant aspects [11].

2.1.2. Documents

Documents are defined as [key, val] tuples, where key is a text identifier field, val is either a value from the available atomic typesets, a document itself, or an enumeration of the aforementioned types. We denote he universe of documents by DOC. This is a traditional way to define documents, which is often provided by JavaScript Object Notation (JSON) or XML.

```
{
  "name" : "John Doe",
  "expertise" : ["C++", "Java"],
  "certifications" : [
    {
      "certification" : "Computer Science Msc.",
      "year" : 2010,
      "by" : "Eotvos Lorand University"
    }, {
      "certification" : "Computer Science Bsc.",
      "year" : 2008,
      "by" : "Eotvos Lorand University"
    }
  ]
}
```

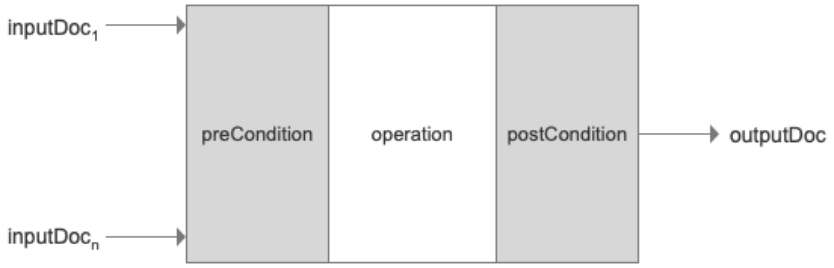
A document schema is defined as [key,typename,multiplicity] tuples, where key is a text identifier field, typename is either an atomic type's identifier, e.g. 'string' or an already defined document schema identifier, multiplicity is a value from set

{1,*}, which denotes if the field is a value, or an enumeration of values from the 'typename' type. We denote the universe of document schemas as DS. A document is based on a document schema, if it contains the fields and values according to the keys, types and multiplicity specified in document schema.[10]

A document schema invariant is a first-order logical expression, where document schema fields referred by selectors are used as variables in the expression. A document satisfies a document schema invariant if the invariant logical expression evaluates to TRUE under the interpretation where the selectors are mapped to the actual document values [12]. A document type is defined as a (ds,inv) pair, where ds and inc is a set of invariants. The universe of document types is denoted by DT [13].

2.1.3. Business Process Workflow Model

The business process workflow is defined by WF: (V,E) directed, acyclic graph where V⊂Activities is the set of vertices, and E⊂VxV is the set of directed edges. In the following discussion vertex and node are used as synonyms. The activities are defined as *[input, output, preCondition, postCondition, operation]* tuples, where the fields are as follows:



Input ⊂ ***DT*** is the set of input document types, defined the following way. The input of v activity is Input={u.output | (u,v)∈E}. So the input is defined as the output documents of all u vertices from which an edge leads to v. Hence only those vertices have input documents that have incoming edges. The vertices that have zero input documents we regard as nodes that produce the output document requiring user input or from database queries. These vertices are called as entering nodes. We introduce the enterNode: WF → ϱ(V) function, which returns the subset of V, that contains all enter nodes and does not contain other vertices. There is exactly one vertex that does not have outgoing edges, which indicates the and of the workflow and is called finishNode. We introduce the finishNode(): WF→V function, which returns the w node that has zero outgoing edges.

Output ∈ ***DT*** is the document type that specifies the result document of the activity.

Each activity has preconditions and postconditions, that define constraints using first-order logic statements on input and output documents.

preCondition: Input → L

postCondition: Output → L

In this study we do not aim to construct a new constraint language, instead we regard these expressions as OCL contraints [14].

Operation: Input → Output function, which processes the input documents and result in one output document, which satisfies the following condition. For a given D set of documents if

preCondition(D)=true, then postCondition(operation(D))=true.

Overall, the graphs' edges indicate the order in which the activities have to be performed and the activities altogether with the edges define the input and output documents for each activity. The universe of such graphs is denoted by ***WFO***. We regard such workflows that they provide a formal specification for implementation.

2.1.4. Client-Server Business Workflow Model

First we introduce a notation into the model, which indicates if a specific activity node has to be performed on the client-side,the server-side or both. For this purpose, we extend the previously introduced model.

The wf2=(V,E,S,C) is a client-server business process workflow graph where (V,E)∈***WFO***, where S⊂V, C⊂V, where all v∈S nodes are performed by the server and all u∈C activities are performed by the client. Note, that S and C are not disjoint by definition, which means that if w∈S and w∈C mean that w can be performed by both layers. The universe of such graphs is denoted by ***WFCS***.

2.1.5. Compatible workflows on client and server-side

The wf2=(V,E,S,C)∈***WFCS*** workflow is called a server-side exclusive workflow, if S=V and C=∅, and called client-side exclusive workflow if C=V and S=∅.

Now we introduce PrS: WFSC → ***WFO*** and PrC: WFSC → ***WFO*** functions, that functions return the server-side and client-side projection of a given WF∈***WFO*** the following way:

WF=(V,E,S,C), WF'=(V',E'),WF'=(V",E")

PrS(WF)=WF', V'=S, E'⊂E |∀ (u,v)∈E':u,v∈S and ∄(u,v)∈E where u,v∈S and (u,v)∉E'.

PrC(WF)=WF", V"=C, E"⊂E |∀ (u,v)∈E":u,v∈C and ∄(u,v)∈E where u,v∈c and (u,v)∉E".

In the simplest case for a given wf=(V,E)∈***WFO***, if a given A⊂V set of activities have to be performed on client-side, simply create a wf1=(V1,E1,S,C)∈WFSC, where V1=V, E1=E, S=V and C=A, then calculate PrC(wf1), which results in the trivial workflow. This workflow is the specification for the client side.

If there are two given business process graphs, wf1=(V1,E1) and wf2=(V2,E2), where wf1 is regarded as the specification of the server-side and wf2 is regarded as the specification of the client-side, we can decide if the wf2 is compatible with wf1.

This is true, if wf1∈***WFO*** can be extended to wf1'=(V1',E1',S1',C1')∈WFSC so that V1'=V1 *bigcup*V2, E1'=E1 *bigcup*E2, S1=V1 and PrC(wf1')=wf2. In this case, the wf2 client-side workflow is compatible with wf1 server-side workflow.

2.1.6. Consistent client-server distributed workflow specifications

So far we did not looked into the relation of workflows and activities. In the previously defined workflow model we defined the start and finish nodes of a given workflow. A wf workflow can be substituted by activity the following way: The input of the new activity is the union of output documents of entering nodes with the according postconditions. The output of the new activity is the output document of the finish node with the according postcondition. The operation is the logic defined by the workflow itself. We introduce the GW:WF→Activity function, where GW(wf)=a, where wf∈ ***WFO*** and a=[in,out,preC,postC,operation], where in=inputNode(wf) out=outputNode(wf) preCV= *bigcup*{v.preCondition - v∈inputNode(wf)} postC=outputNode(wf)preCondition,task

If GW(wf)=a for a given wf∈ ***WFO***, then the workflow is consistent with the activity and the workflow is the refinement of the a activity. We introduce RF:Activity→ϱ(***WFO***) function, where RF(a)={wf| wf ∈ ***WFO*** and a=GW(wf)}.

If wf=(V,E,S,C):∈WFSC, and we select a subset A⊂C, and we substitute each a∈A with a wfs∈RF(a), resulting in wf'∈WFSC, then PrC(wf') is consistent with PrS(wf). We can take also evaluate if a given wf1=(V1,E1) specification is available for the server-side and a wf2=(V2,E2) specification is available for the client-side if wf2 is consistent with wf1. The wf2 is consistent with wf1, if there is a wf3 graph that is compatible with wf1 and wf2 can be produced by a finite number of refinements from wf3.

3. Summary

Definitions and procedures described above will result in a formal tool that enables the analysis of the consistency of a particular business logic both on the server and client-side. The tools introduced will allow us to split a process specification into two specifications for both the server-side and the client-side. In other terms, a formal verification tool is given to decide whether the two separate specifications for the server and client-side are consistent. Based on the specification, automated unit tests can be generated from preconditions and postconditions, which help to verify the new program.

3.1. Further work

The above construction gives vent to further research. The next step would be the construction of an algorithm that would check the compliance and compatibility of separated processes as it is given in definitions 3.2.1 and 3.4.2 which can be later implemented. Further transformations of the business logic can be analyzed, which are applicable without damaging the consistency. Such a transformation type can be the introduction of processes and activities which can be executed in ad hoc order. Another question is whether the model is suitable for generalization for different architectures apart from client-server. Microservice architecture seems to be an obvious choice where processes would be distributed among microservices.

References

[1] Berners-Lee, Tim and Cailliau, Robert and Luotonen, Ari and Nielsen, Henrik Frystyk and Secret, Arthur: The world-wide web., *Communications of the ACM* 37.8, 1994: 76-82.

[2] Garrett, Jesse James: Ajax: A new approach to web applications. *Adaptive Path*, 2007.

[3] Graziotin, Daniel and Pekka Abrahamsson: Making sense out of a jungle of JavaScript frameworks, *International Conference on Product Focused Software Process Improvement*, Springer, Berlin, Heidelberg, 2013.

[4] Jain, Nilesh and Ashok Bhansali and Deepak Mehta: AngularJS: A modern MVC framework in JavaScript, *Journal of Global Research in Computer Science*, 5.12, 2015: 17-23.

[5] Staff, CACM: React: Facebook's functional turn on writing Javascript, *Communications of the ACM*, 59.12, 2016: 56-62.

[6] Wohlgethan, Eric: Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js. *Diss. Hochschule für Angewandte Wissenschaften*, Hamburg, 2018.

[7] Máriás, Zsigmond and Tarcsi, Ádám and Nikovits, Tibor and Halassy, Zoltán: Benchmark-based Optimization of Computational Capacity Distribution in a Client-Server Web Application *Acta Electrotechnica et Informatica*, Vol. 13, No. 4, 2013, 79–84, DOI: 10.15546/aeei-2013-0053.

[8] Kurg, Steve: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516, 2014.

[9] OMG Unified Modeling Language, Superstructure Version 2.2, 2008.

[10] Molnár, Bálint: Applications of Hypergraphs in Informatics: A Survey and Opportunities for Research, *Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae Sectio Computatorica*,vol.42, 2014, 261-282.

[11] Molnár, Bálint and Máriás, Zsigmond and Suhajda, Zoltán and Fekete, István: AMNIS - Design and Implementation of an Adaptive Workflow Management System *9th International Symposium on Applied Informatics and Related Areas - AIS2014.*

[12] Molnár, Bálint, and Benczúr, András: Facet of modeling web information systems from a document-centric view. *International Journal of Web Portals (IJWP)*, 5(4), 57-70, 2013.

[13] Molnár, Bálint, and Benczúr, András: Modeling information systems from the viewpoint of active documents. *Vietnam Journal of Computer Science*, 2(4), 229-241, 2015.

[14] Kleppe, Anneke and Warmer, Jos and Cook, Steve: Informal Formality? The Object Constraint Language and Its Application in the UML Metamodel. In: Bézivin J., Muller PA. (eds) The Unified Modeling Language. «UML»'98: Beyond the Notation. UML, 1998. *Lecture Notes in Computer Science, vol 1618.* Springer, Berlin, Heidelberg.