2024.03.22 Meeting notes

Date

22 Mar 2024
Time: 16:00 (CET)

Meeting link https://tib-eu.webex.com/meet/georgy.litvinov

Attendees

- Georgy Litvinov
- William Welling
- Dragan Ivanovic
- Brian Lowe
- Mark Vanin
- Miloš Popovi
- Ivan Mrsulja

Agenda

- Published updated Dynamic API branches
    - https://github.com/vivo-project/Vitro/tree/dynapi-1.14.1-snapshot
    - https://github.com/vivo-project/VIVO/tree/dynapi-1.14.1-snapshot
- Check style alignment
- Fixed report generator endpoints method
- Fixed binary file conversion in json converter
- Standard license headings added to java files
- Cleaned n3 files
- License heading has to be added to n3 files
- Configuration bean loader modification
    - Added implemented by object property
    - Fixed dynamic api classes and tests to align with configuration bean loader modification
    - Updated dynamic api class to implementation binding
    - Align configuration bean loader modifications with main branch https://github.com/vivo-project/Vitro/pull/446
    - Created new configuration bean loader tests
    - https://github.com/vivo-project/VIVO/issues/3109
    - Tests to be added
- Authorization policies to support dynamic API to be created
- Array implementation using JsonContainer
    - Additional fixes are needed:
        - Refactoring of JsonContainer related classes

Meeting notes (transcribed automatically)

Georgy: Okay then we can slowly start and maybe he'll come and maybe Brian will come. We'll see. Okay so as always this meeting will be audio recorded and we will have some transcribed meeting notes.
Feel free to fix them if they're not correct. I don't know if you have looked at some previous meeting notes. We can't hear you, Ivan. Some issues with audio, I suppose. Okay. Let's, okay.
So I don't know if you looked at today's agenda. So first thing is that these dynamic API branches on the top of the latest VITRO and VIVO modifications were published in VITRO, in view project VITRO and view project VIVO.
And there are a few commits to do that, to rebase that. And we will get to that a little bit later. So first big thing that was done is, I don't know if you remember that in the development of 1.14 VIVO main branch,
This check style was enabled by default, so all the Java files not excluded, not specifically excluded, should comply with check style rules and I fixed check style validation errors that we have in dynamic API for dynamic API specific classes.
So there are a lot of fixes, a lot of changes, but that's all just styling, refactoring, so no real code was modified there. So the next point is about the method for repo generator endpoints.
I think Mark will talk about that right before your vacation.

Mark: Yeah, so I'm currently as well, uh, glancing at, uh, commits and I see like you have changed the default methods from boot to post and some other ones regarding reports and writers. So I will look at this one later.

Georgy: Yes. Now that should work without any, anything else right now. Yes, also, as you remember, right, I think on Tuesday, right before you get to a vacation, I found this mistake that was in JSON converter where binary files
were converted back and I changed it, I fixed it. It's not a boolean, of course. It's serialized as a text note and yeah, that should work. So, the next point in the agenda is about the standard license hearings, for some reason,
I think in some files we forgot to do that. It's done in all other, I hope, all other VIVO Java classes that we have in VIVO and VITRO projects. So that just hitings that I added with the same pattern that we have in VIVO and VITRO.
So once again, a lot of files, but nothing really changed here. So also, and that may be interesting to you. So I used some prefixes to make current entry file a little bit more readable. So instead of having these full URIs everywhere,
I created prefixes, then in KPI HTTP methods, for example, here you can see implementation type, serialization config. And I think that way it's much more readable. And also in case some URI prefix is changed, it's much easier to do that, to fix that.
So you see it's everywhere here. So for RDF types, so all the files were a little bit improved. So getting... more so I didn't do that, but I plan to add the license heating the same way that existing in the Java files and they also exist in some
VIVO and VITRO files. So we need to have license heatings there just to specify the license of the VIVO project that we use. And yeah, that's not done yet, but it's going to be done hopefully soon. Also...

Mark: Yeah, I can see that there have been lots of improvements and changes in those two weeks, like around five to six pages, like small ones, mostly commits. But so, yeah, there have been lots of job done.

Georgy: Yes, and that's if you go for example to look at this VITRO branches and we go to the commits. So just to explain to you how this rebase worked from 1.14 that was a release 1.14.0. to current 1.14.1 snapshot.
So I created at the start of commits, I created a new commit that reverts all the change made in VIVO main branch and created commit after at the end of all of our chain of the comments that puts that back and resolves merge conflicts that we have.
And also, so maybe I'll show it to you. So it was, I think, somewhere here. So there was a compile error after rebase, and that's the commit to restore this change. So we had merge conflicts for model names, configuration triple source,
configuration model setup, and startup listeners. So not much, so that was more or less easy. Compilers, because we have changed this method, applyChangeToWriteModel, so that all the change from custom entry forms go through writing with a RDF service,
which is able to notify all the listeners about this change. And not only that, but also it it's able to provide the URI of the user so that in the model that called audit so that's tool to track changes in VivoGraphs we would be able to see these modifications
and so yeah I had to wrap it here with a dev service model that's not not ideal, because a dev service model is just a wrapper for a model to be able to work with a dev service. Usually that should be avoided, because that's not performant if the model is
initially from a RDF service and this model should be accessed a different way, but as a temporary solution, until we resolve these difficulties with the RDF services, that should work. And that's not very bad right now, but this is something that has to be
revisited and fixed one day. Yeah, so that was the complication about this rebasement. Sorry, I just wanted to give you more information about that. And one of the big problems and maybe you remember if you were on the last meeting was about configuration
bean loader. And if you can see here, so I created a few commits to modify configuration bean loader. I don't know if you... So there are two files, two Java files, configuration RDF parser and configuration bean loader.
And configuration bean loader basically the service that creates a Java instances from triple source. So it reads the data and creates a Java instances and create and is able to provide. So it's basically dependency injection for semantic data.
So we are able to create big classes with arbitrary complexity by asking this configuration bean loader to load some individual. And this configuration bean loader is following all the dependencies in classes and creating the instances.
And also, so one way to do that is to provide an instance to that. And then another way is to provide a type of a class so that the configuration bean loader would find all the real instances that should be loaded.
For example, we have procedures, as you know, in Dynamic API. And to load all the procedures in procedure pool, we need to find all the procedures first. So that's done by configuration bean loader.
So it finds all the instances and then load them one by one. And we had a problem that previous implementation, so initial implementation done by Jim Blake, it loaded data it loaded the triples but
these triples should be specified as a type with the resource and the URI of this resource should be Java then the path or over the Java class and initially I created Java implementation.
Maybe I'll show it to you here. Initially, I created the binding between the ontology classes and implementation classes. Hi, Dragan.

Dragan: Hello, hello.

Georgy: Initially, the binding between ontology classes and implementation classes was done with RDF subclass of object property. But this object property is a transitive and it is used by the reasoner, by the standard ontology reasoner
that's provided in Jena, in ont model, for example, if you use ont model. And the result of it is that you will have for each individual, you will have all the... If it has a hierarchy of classes behind it, then you will have all the implementation types here.
And then configuration bean loader is confused because it has two main implementations to load one individual and it can decide about it. So the problem was to provide information for configuration bean loader to decide that.
And yeah, and Dragan last week, we discussed that and Dragan, for example, proposed a solution to be able to define the hierarchy of that implementations in the same file. So in basically in this file, so we have this implementation.
So that's a URI of implementation and to provide this way. And yeah, that's a good solution, but I wanted to get a solution where we can fully decouple ontology part of dynamic API and the implementation part.
And if you have to define the hierarchy of Java implementations on the Ontology side, so in the triple store, then people who later working with Ontology would have to know something and update information if the implementation part changes.
So for that reason, I found another solution. I created dynamic API object property. So you see it here, it's dynamic API implemented by. And this implemented by property differs from this subclass because it's not the transitive property and it's not used by the
so you won't have inferred statements. So that's only one part of the problem. So problem with hierarchy on the side of Java implementations, because in Java implementations, we can use inheritance and we would like not to have a constraint on that.
So the other part of that problem. So here you see the query that finds finds the instances by Java implementations. So in this query, this is the old part, how it was done. It was not done in this particular query.
I just implemented it so that would be a fallback for standard we were loading because we will load this file, for example, to load the application itself in configuration file application 3.
And this is the way, so by just specifying that we are looking for URIs that has type and this type is Java URI, specified it here. So we are able to make this function work for both for standard view,
how it was implemented before, and for dynamic API specifics. I think it's a broader concept, I would say. And here you see that this is the function, so find URIs that was affected.
And another one and maybe a little bit more complicated is construct query that I created to be able to find implementations for some individual. And it also has a fallback, so you see there is a union and the top left side of that union is a fallback,
so we find all the types of some individual and filter that they should start with Java. And the right side of this is a way to find out which implementation we would like to use because we could have inheritance and more than one ancestor in that
inheritance might have set this Java implementation object property. And to do that, I use these sparkle path how it's called path property, this asterisk you see. So this asterisk means that we are looking for a subclass and it could be not just
the subject in this property, but it can be indefinitely away from that. And it's also can be so the type also can be interpreted as intermediate class because asterisk also means no, so zero distance.
And the same is done on, so we find these intermediate classes and that's the final class that has this implemented by. And we find the distance between our individual and some ontology class with Java implementation.
And then it's here I group it and I get the distance for each implementation to be able to range. So the fallback has zero distance, of course, but all the dynamic API ways to define that have more than zero instance.
Of course, this query, it's broader, so it can find in a tree-like structures, but of course if you have two, because in ontologies you are able to define more than one, so inherit from more than one class, and if these classes
inheritance chains have some the same ascendant then after that ascendant and on this ascendant it will have wrong priorities but I think that's fair enough for our use cases. So I don't think that we will use something like that
and maybe by that time if we get to that then we would be able to use maybe more full featured sparkle constructs that easily allow us to do so.

Dragan: So basically, Georgy, you implemented that in both directions, right? So if I have the name of the anthology class, I can get the Java implementation of Java Bin for that. That's the second you presented.
And the previous one you presented, it was for the another direction, right?

Georgy: So if we have a... Yeah. First one, if you have a Java implementation name, so class name, then you can find all the individuals. And this one, this one, is to find all the Java implementations by individual.
So for this one we need to find all the implementations and here and that's optimized and limited version so it will provide us for example, results with 0 distance, with distance 1, with distance 2, with distance 3.
And I do not limit it here. I have a query that is able in Sparkle to get only the distance 0 distance. But it's much less performant because it's sparkly. It's just not performant at all.
So for that reasons you would see here that after we have these solutions with Java implementation name and the distance, we use only the lowest distance. So the first it's sorted with ascending order.
So the zero will be first the one or two but all of them are more than zero and if we get to a solution that has a different distance so we have maybe two with zero distance then we just go out because we are not interested
we already have two solutions that we are going to use. And only if there are two solutions with the same distance, this will result in the same... How it's called... There was an exception where this... Yeah, I think it's here.
When concrete classes are more than one, then we throw these too many classes exception. But as you can see, a few exceptions will not be thrown. I think some of them weren't used at all. But exceptions like is it a URI.
I'm just checking most of that in the query. And if not, then I should do that. And also, I think something else was removed. Yeah, that's confirm eligibility for result class, because I think it checks basically absolutely the same thing
that does other code and it checks it in wrongly. So, yeah, that was removed and that's not needed. So, that was about this configuration with loader. And yeah, as I said, maybe here I created this object property that's used there.
I also use this. So I fixed dynamic API classes. So dynamic API classes would work with that because we also coupled with that implementation. And for that reason, for example, I removed this check is it in model for the abstract pool.
Because I think that maybe I suppose abstract pool should not check that. So it should not be its responsibilities, just too much coupling between the pool and the models that it might use. Because we would plan to use different models,
so different graphs for different data, it doesn't really make sense to have that and re-implement that to check the same thing, but maybe not with these object properties, but with the sparkle queries, so that was removed.
And yeah, and this one I will show that's fixed for implementation binding, this one. So everywhere just subclasses were replaced by implemented by and also I think I don't need to provide interfaces anymore because it's just not needed.
We can infer that from the concrete class in Java. Yeah. Also, Dragan, if you find time, please review and check that pull request that I created. I don't know.

Dragan: This was a draft pull request at one moment, right? And now you switch that to be ready for you, right?

Georgy: One second. Yeah. Yeah. Two weeks ago that was marked as ready for review on 8th of March and it was almost at the start. And that changed only that I cherry-picked these fixes for the Dynamic API branch alignment.
So what was already in Dynamic API? And today I also cherry-picked this gamete that I was talking about right now. So if you find time, please check that. Also, I added some tests because we already had this,
how it's called, load as URI. Load as URI, what's it called? Annotation, yeah. And load integers and load boolean, so now it's tested. We had that functionality in dynamic API, but it wasn't covered with tests.
Maybe if I find time, I'll add maybe more tests because that's my idea. And that can be added as a new pull request. So feel free to review that because the faster it's merged and we align that with the main branch, I think the better.

Dragan: Yeah, sure.

Georgy: Yes, so what's next? Yeah, that's what I was talking about. So that's the test. So there is a few tests and modification of the test data. Also, yeah, be that I didn't say that alongside with that modification of configuration
bean loader the test suite takes a little bit more time. So it was 4 minutes before that and now it's 4 minutes and 40 seconds. So something like that. I tried to optimize it as much as possible because initially that was 8 minutes.
Yeah, but I don't think I can do it even faster. Most of the time in test takes for initial initialization for the tests for each class of the test. And yeah, that's something that I don't think we can make faster at least right now.
Also, I would like to, while working with that, I would like to point you to this VIVO issue that we have. There is a discussion, there was a discussion between Mike Conlon and Andrew Woods about this modification, VIN configuration bean loader.
I don't know if you're familiar with that, but this configuration bean loader, is able to load data with different prefixes. Maybe I can show you that. So, yeah. So, for example, if in some file we have this prefix that looks like this,
then configuration bean loader will be able to load that because what it does for each full stop or dot, it replaces this dot with a hashtag and tries to load a class with that, tries to find this implementation in a triple store.
And as you see, so the more, the longer your name of the class or the full name of the class, the more attempts it will try there will be more sparkle queries that I may be able to show you the code that related to that.
So I think it's, yeah, it was somewhere in configuration RDF parser, I suppose. One second, I'll open it to you.

Dragan: Georgy, the issue is that any package requires some action, right?

Georgy: The issue is that I think that it's not necessary to try to... No, it's configuration. One second. It's not necessary to really do that, to really try to find that. So this is a canonical URI and it's used. Yeah, it's used.
Yeah, that's I think. So it finds all the possible URIs. So for example, if you provide the configuration bean loader Java URI, it will still split your URI into pieces, that's URI pieces. And for each numbers of the dots in your URI,
it will try, it will produce a new URI with hashtag that replaces one of the dots just it's done for I suppose it's done to make it more because of I don't know ontologists friendly how to call it because if we go to the VITRO or
VIVO main configuration file and we open that configuration file so that's the application setup entry and you see that here you see this java URI and it ends with hashtag so that people could use hashtags here and for example close it
and using URI something like that, oh sorry, something like web app application. I don't think we need that. So much easier thing to support and to work with for configuration bean loader, not to spend time to find, uh,
any possible thing in a triple store is just always have this if we need to define a prefix that ended with dot it's valid. Maybe it's not. It not confirms maybe some practices because usually you answer prefixes with slashes or hashtag.
But it's a different case. We have this Java implementation URI, so it's very specific case, and it's still valid syntax to do it that way. So that's why in this issue, I think Mikle Kornel said that the prefixes it's a shortcut and the string
following the prefix as the text to substitute it for the prefix. So it's basically a thing to replace before parsing URIs. It's nothing specific and we should not do it this way. So like it was done. I think, yeah, here it was...
Sorry, I won't find it. I had a link to that, but I don't remember where it is. So, yeah, that would be much more performant, I suppose, because we traverse the triple store a few times just to make it more beautiful. It doesn't make much sense, I think.
And also, yeah, I wrote it in an agenda that I plan to add maybe more tests for dynamic API specific tests. We'll see how it goes. And also I can say that I wasn't able to work on authorization policies yet to support dynamic API.
As you might remember, last time we discussed that there should be policies that could support authorization, some simple authorization settings that we could be able to use in dynamic API procedures. Yeah, also there was this array implementation
using JSON container, and I merged that, not merged that, I just cherry-picked the code. But it's still not finished. I can say that it more or less work. But we need to refactor this JSON container and to heavily modify that.
But in case you're interesting, yeah, for example, that's the use case even gave me. And by using this use case, for example, if I edit one and use it here and then go to a person's and go here. We see that edit this individual and show raw
statements with resources as a subject. We see that given name has two literals. So that works, but it's not in the comments because it's not ready for production code.

Dragan: Is this going to be possible in another direction, meaning when I'm fetching the person to get in the JSON also the array of those?

Georgy: Yes, yes, of course. But the current problem is that just the implementation of working with the JSON data is not ready at all. And after the modification that I described to you about this configuration bean loader, we don't have this
constraint between the ontology and Java implementation. And because we don't have it anymore, I'm going to refactor the code a lot. I hope that it wouldn't be significant, but... it will make the Java classes more reusable and less bad code.
I don't know how to call it. So then it will be usable. Now I just created a small fix in my IDE and the fix works and it should work both ways. I would say that the second way, I think it's covered with the tests, with the pull request
that I told you about some minutes ago. But the main reason right now that I need to maybe couple the use cases. So now we have these JSON containers, but JSON containers are used not only like, so you can just parse JSON and then you have
parsed JSON in your instance object. We use Jackson library for that. I think it was the most useful and the most feature rich. And the problem is involves that we also would like to have some methods to easily get something from array
and put something to an JSON array and the same way work with objects. So we created some methods that able to store real implementation of objects and how now it works. It saves keys as node values in JSON and then it's the real
real objects in memory are stored nearby in a hashmap and when the key is requested then it's found from the node value and the value is used as a key in a hashmap and then the real object is returned. Now it's a little bit mixed as you might understand.
So their responsibilities is not only to parse the JSON, input JSON, but it also can be used, for example, to put something real, some already instantiated object, not to do the same serialization, deserialization multiple times during procedural execution.
Yeah, and for that reason it has to be refactored. We'll see, I'll let you know how it's going. I've already started this process. Yeah, so...

Dragan: Yeah, but we already did refactoring. Is that going to help us to have, let's say, the full control over the needed JSON input and the expected JSON output?

Georgy: Yeah, it will allow us, of course, to have functions that work specifically with arrays, specifically with objects, and we're already able to produce some objects and arrays and combine them and create some complicated and maybe complex output,
JSON output, but the problem is that there is no specific high level library that is able to help you map JSONs easily. And we can just reuse it. So yeah, there are something, but not something that can be easily reusable sometimes because of dependencies,
sometimes because of the license limitations. So right now it's just create an array, put something in an array, and the same with the objects. So that simple modifications are easy, but if you compare that with something that I did for GESAH project,
And if you give me a second, I'll show you that what I'm talking about. So here, this is a repository that I'm working with. And in this repository, dynamic API is used. And if we go to the dynamic API, A box. You see there is nothing complicated,
so algorithm is very basic, so we have a few steps and it checks the presence of something and then it does basically two things. It transform, it makes a sparkle construct query. So I think this is the operation export culture object to Lido.
So that's a sparkle construct query. Then it does the XML transformation. And after XML transformation, it does XML validation via the XML schema. And you don't see the XML schema here, of course, and XML transformation. So that's in other files.
And that's just some parameters that I use here. But if we go to the real XML transformations that exist here, so that's the file and that's just a parameter defined here as a string. And that's the mapping itself. And that's not a small mapping, as you see.
And to do that, so I had to write this file. Yeah, look at that if you are interested and maybe you'll find some mistakes. But this API, so we have this Lido export and this Lido is already used and it's already even public on, I think it's called, one second,
I'll find it, on a graphic portal here. So that's the data from GESAH that was imported here. And yeah, there is a lot of, I think something like...

Dragan: And this is imported using the Lido format.

Georgy: Yes, that's all the imported using a Lido format. So maybe in some long... Yeah, we just click here, and so all these 2636 cultural objects are imported with this function. Here. And for each one you can see some images,
some data, and you can go back to the real to the real description of that object and look at maybe even better quality of image because they have limitations in their graphic portal on the image quality. Yeah, so if we compare to that,
so that's not ready and most likely it will require, if you're thinking about this JSON conversion, that it might be much more complicated than it is for the XML. And there is, for example, what I'm also planning to do for this JSON is
finalize maybe validation of JSON because we are able to validate the input JSON and output JSON and at least some basic functionality is there, but I don't remember if that's enforced, most likely not. And yeah, that's something to be
finished and also be the latest updates in VIVO. We don't need a select context for many of our classes anymore. So before, if you're not familiar with the issue, all the access to the graphs, it was stored on the context, a select context,
and to be able to access graphs in runtime, you had to initialize these instances with a select context so that it would be able to queries or add some data into the graphs. And now that's converted into the singletons. So there is no need to do that.
And I already simplified some code in dynamic API and I'm going to continue to do that. Yeah. Do you have any questions and remarks? It's very welcomed.

Dragan: Yeah, it is, but it's not so easy to consume everything you informed us, basically.

Georgy: Yeah, but overall we have, I think, some good progress. And I expect that after 1.15 is released, our shore to rebase on top of 1.15 will be minimal.

Dragan: Yeah, and you said it's quite important for the main branch to merge the configuration bean loader improvements and that's right?

Georgy: Yes, I would like to have it synchronized before we somehow find some other problem in configuration bean loader and then we will have diverse branches related to this configuration bean loader and yeah, that would be a problem.
It's not something very urgent, but as it's more or less safe operation and I would like, I would try to add maybe more tests. I would like to see that merged as soon as possible, yeah. But nothing like, it's not blocking anything at the moment.
It just plans for the future.

Dragan: Yeah, but it should be part for sure of 1.15. Yeah.

Georgy: Yeah. Yeah. And we talked about the test coverage. I hope we will improve that. With that too. Because I think for dynamic API, we we cover more than it's done in VIVO at least. Hopefully this will be improved more because what I've noticed
that is I covered, for example, I think JSON converter maybe not covered at all or covered insufficiently because I, because a lot of bugs were found in that and bugs, I mean, just simple like this simple, maybe typos or just something that I overlook.
So that should be covered too. And yeah, we are getting better and better. And the next would be, I suppose, to resolve the issues after refactoring, resolve the issues where RDF service and we'll see if that will be synchronized with the update of Jena.
I'm curious about that because our RDF service, maybe I told you, Dragan, that we have... Yeah, I think yesterday we talked about that we have... more than one way to access graphs and one, the good way to do that is by using RDF service and the bad one
is by using these ontology model selectors. But we can't at this point just get away from ontology model selectors because there is a caching that's done outside of RDF service. And this caching is in memory models. So every model that's initially stored
in configuration triple store is cached and also the models T-box. So the model related to ontologies that is stored in a content model is also cached. So yeah, we need, I think we, I had a conversation with Brian sometime ago that maybe we need to to move
this caching into RDF service implementation, but most likely we need to even make it configurable because now every model is, and every graph is somewhat unique and it's very hard to maintain and it's very hard to, it's not a flexible configuration that we have.
It should be more flexible. Okay, so that was all I had for today. Do you have anything else? No? Okay. So, have a nice weekend.

Dragan: Thank you, Georgy. Thank you for a nice presentation.

Georgy: Thank you.Thank you.Have a nice weekend. Have a nice day. Bye-bye.

Ivan: You too. Bye-bye.

Mark: Have a nice week.